1987

Implementing Path Control, English Motion Commands and Off Line Programming on Minimover-5 Robot

Mark S. Chen
University of Central Florida

IMPLEMENTING PATH CONTROL, ENGLISH MOTION COMMANDS AND OFF-LINE PROGRAMMING ON MINIMOVER-5 ROBOT

BY

MARK SHI HAU CHEN
B.S.I.M.E., Escola Engenharia Industrial-Brazil, 1984

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1987

ABSTRACT

According to Alan L. Porter and Frederick A. Rossini (1987), 50,000 robots are expected to be installed by year 2000. This increasing number of robots to be used in industry has lead to the need for skilled robot operators and programmers.

However, to train this specific class of personnel using an industrial robot as a training tool can be expensive and dangerous due to the possibility of robot accidents. Therefore, in order to provide a safe, inexpensive, and effective training tool, the use of the MiniMover-5 educational robot is proposed.

The MiniMover-5 has only a primitive programming capability. A user-friendly software package could accelerate the learning process. The developed software provides an improved programming approach, over the original MiniMover-5 programming method. It does this by incorporating english motion command structure, path control and off-line programming capability.

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. John E. Biegel for his advisement not only on this research but also throughout my graduate program at University of Central Florida. I would also like to thank Dr. Chin Lee and Dr. Abmad Elshennawy for their help in the preparation of this thesis.

In addition, I am grateful to Ms. Jenifer Sargeant for her great help during the writing process of this thesis.

TABLE OF CONTENTS

Appendix

TABLE OF FIGURES

CHAPTER I

INTRODUCTION

In today's fast-growing field of automated manufacturing, industrial robots are becoming increasingly important. According to Alan L. Porter and Frederick A. Rossini (1987), 50,000 robots are expected to be installed by 2000. This projection resulted from to several factors that included:

1- Public awareness of robot technology and its potential application.
2- Robot technology improvement.
3- Robot unit price reduction. (It is expected a 2% annual robot cost reduction, thereby making robot application projects easier to justify).
4- Expansion of the robotics market beyond the large corporations.

The increasing number of robots to be used in industry has lead to the need for skilled robot operators and programmers. In order to satisfy the industrial robotic personnel needs, a specific class of personnel

must be trained. These personnel can acquire their robotic skills through the technical institutions, such as universities and vocational schools.

Universities can provide a theoretical background while vocational schools can provide a technical background. Hands-on experience can be acquired through laboratory assignments.

Safety hazards should be considered when using an industrial robot as a learning tool. Robot accidents may result in human injuries, robot damage, lawsuits or other legal liabilities.

Educational robots can be a solution to this problem. Educational robots can be used as a learning tool to meet the need of safely training students while maintaining the proper "industrial" working environment.

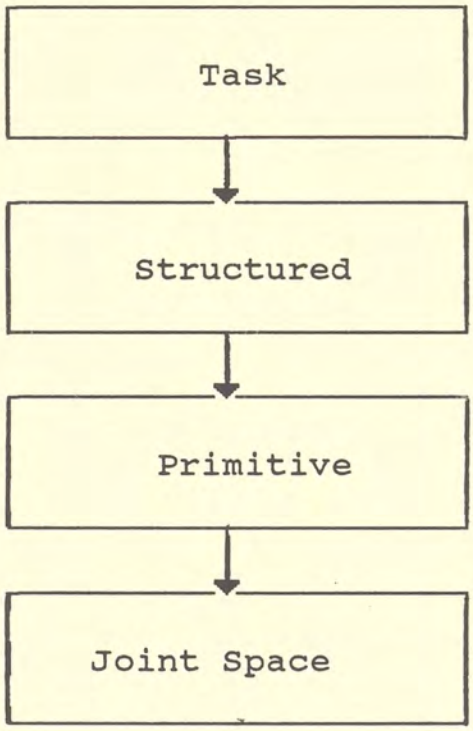
The Programming Languages

The basic communication mechanisms between human beings and robots are the programming mechanisms.

Early robots were controlled by limit switches which provided little programming flexibility. Many present-day robots are controlled by digital computers,

some with multiple processors which permit a high level of user and machine communication.

Robot programming languages can be classified according to the level of interaction between the user and the system during the programming process. According to James Rehg (1985), robot languages can be classified as shown below:



Joint-space control languages concentrate on the physical control of robot motion in terms of joint and

axis coordinates. The joint-space control task is to convert the desired joint coordinates into the appropriate signal to control the force and torque of the motors. This type of language is hardware dependent. The advantage of this level of control language is the low cost of software development. The disadvantages include the lack of integrated work cell commands required for system control, the absence of any cartesian coordinate values for programmed points, and severe limitations on path and velocity control. Examples of this type of language are found most frequently on the many educational robots such as Armbasic used with the MiniMover-5 educational robot and the Rasp language used with Rhino XR educational robot.

Point-to-point primitive languages are the most common type used on the industrial robots available today. This type of language consists of leading the robot through desired motions and recording the positions and orientations of the robot. The user can create a sequence of steps that can be played back any number of times. The advantages of this type of language are that it is simple to use, easy to debug, and commercial packages are readily available. The disadvantages are little programming

capability, emphasis on the robot motion rather than on the task to be performed, and non-existence of expansibility of off-line programming.

A structured programming language is basically an inproved primitive programming language. The major advantages of the structured programming languages over the primitive motion level lies in the availability of complex data types, structured programming format, complete control structures and the feasibility of off-line programming.

In the previous programming methods the task is described by the user through a series of robot motions. For task planning languages, the user gives the final state of the task and the desired operations on the specified object. The system then decides on how to achieve the final goal. Once the intermediate steps are defined, the system converts these specifications into the robot-level specifications. Languages at this level are not commercially available, but they are currently in the development stage.

The Definition of the Problem

The problems previously mentioned, that of using an industrial robot as a learning tool, show the need for other alternatives to train the beginners in the robotic field. One alternative is the use of educational robots which can emulate industrial robots.

The robot used in this research, the MiniMover-5, is an educational robot from Microbot Inc. The MiniMover-5 is interfaced with the Apple Plus personal computer through a Microbot Firmcard (Microbot Inc. 1981). The communication between the MiniMover-5 and Apple Plus PC is done through ArmBasic language (Microbot Inc. 1981).

The ArmBasic language falls into the joint space category. In order to move the robot arm to a particular location, the user must specify each of the joint angle increments. This type of programming is difficult because industrial tasks normally have a large amount of information to be programmed. The user is susceptible to a high probability of committing programming errors.

Objective

The research discussed herein is to develop a friendly software package that will raise the level of the MiniMover-5 programming language, to provide path control and off-line programming capability for the MiniMover-5. For the purpose of this research, off-line programming is defined as the capability of programming the MiniMover-5 robot without having the using of the robot during the programming process.

The result of this research is to develop a software package that enables the user to simulate the programming of an industrial robot.

The software will have the equivalent MOVE, MOVES, APPROACH, APPROACHS, DEPART, DEPARTS, OPEN and CLOSE commands of VAL II language. The software emulates the VAL II motion commands and will be written in the Apple's Basic language. The user will be able to simulate an industrial robot programming task through an edit mode which will also be developed using Apple's Basic language.

The Importance of this Research

This research will provide motion control software in addition to an editing capability within the MiniMover-5 robot. This will result in:

1- Improvement of the man-MiniMover-5 robot interaction.

2- Improve the MiniMover-5 programming capability.

3- Provide the MiniMover-5 with a path control capability.

4- Reduce the probability of user programming errors.

5- Provide an off-line programming capability.

CHAPTER II

THE HARDWARE

The Mechanical Construction

The MiniMover-5 is a revolute robot with 6 degrees of freedom. The MiniMover-5's performance characteristics are shown in Table 1 (Appendix A).

The major components of the manipulator are shown in Figure 1. The manipulator consists of a fixed base within which the computer interface card is housed. The base joint connects the body to the base through a hollow shaft. The upper arm is attached to the upper end of the body by a pin parallel to the base. This is called shoulder joint. Similarly the elbow joint connects the forearm to the upper arm. The wrist joint connects the hand to the forearm. The hand joints control the grip.

Each of the joints are controlled by a stepper motor which drives separate cable drums by means of individual reduction gears. See Figure 2. Motor 1 controls the base joint causing the rotation of the body about a vertical axis through the base. The body can rotate ± 90

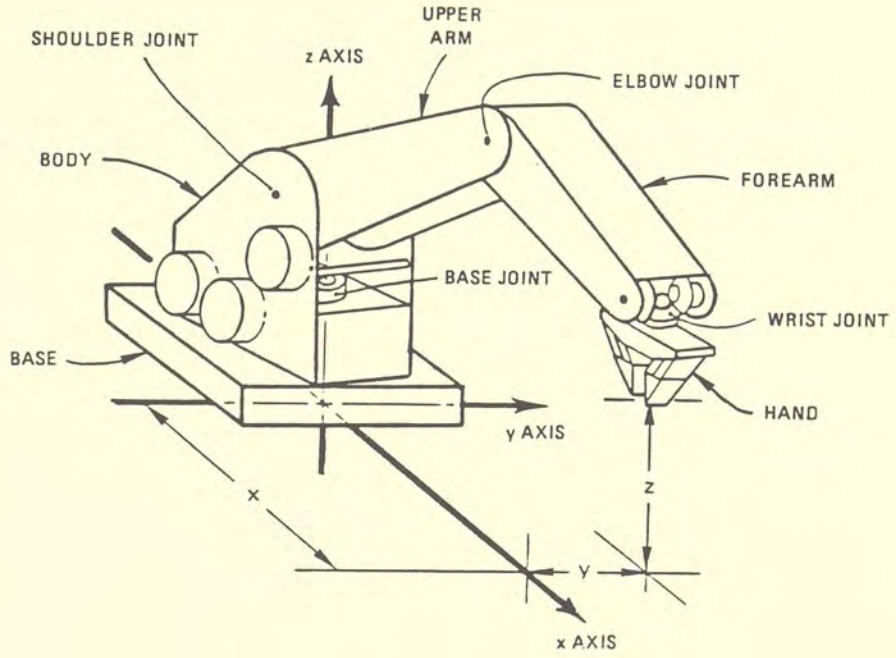


Figure 1. Major Structural Components
(Microbot Inc., 1981)

degrees. Motor 2 causes the rotation of the upper arm about the horizontal shoulder axis. The upper arm can rotate through +144 to -35 degrees from horizontal. Motor 3 controls the elbow joint causing the rotation of the forearm about the horizontal elbow axis which connects the forearm to the upper arm. The forearm can rotate through 0 to -149 degrees from the horizontal. The wrist joint, consisting of right and left wrist (see Figure 3), is controlled by motors 4 and 5 respectively. These motors

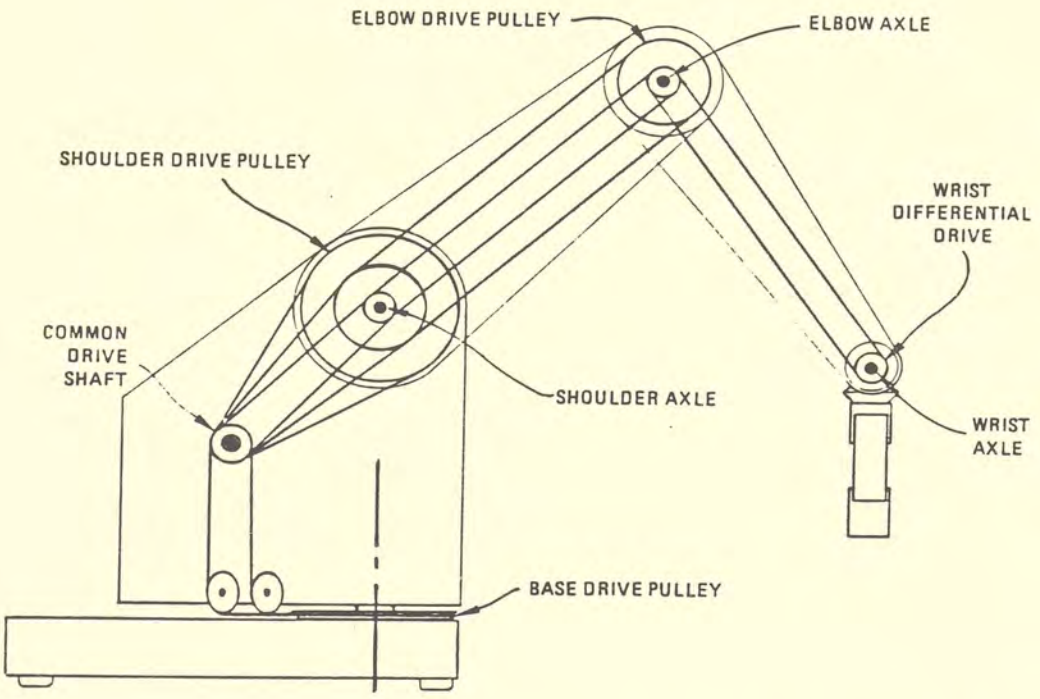


Figure 2. Cable Drive System (Microbot Inc., 1981)

cause the hand to roll and pitch relative to the forearm. Finally motor 6 controls the opening and closing of the hand.

The Stepper Motor Control

Each of the cable drives is controlled by a stepper motor. The stepper motor consists of four coils, each driven by a power transistor. The transistor can assume an ON or OFF status. A particular combination of transistor's

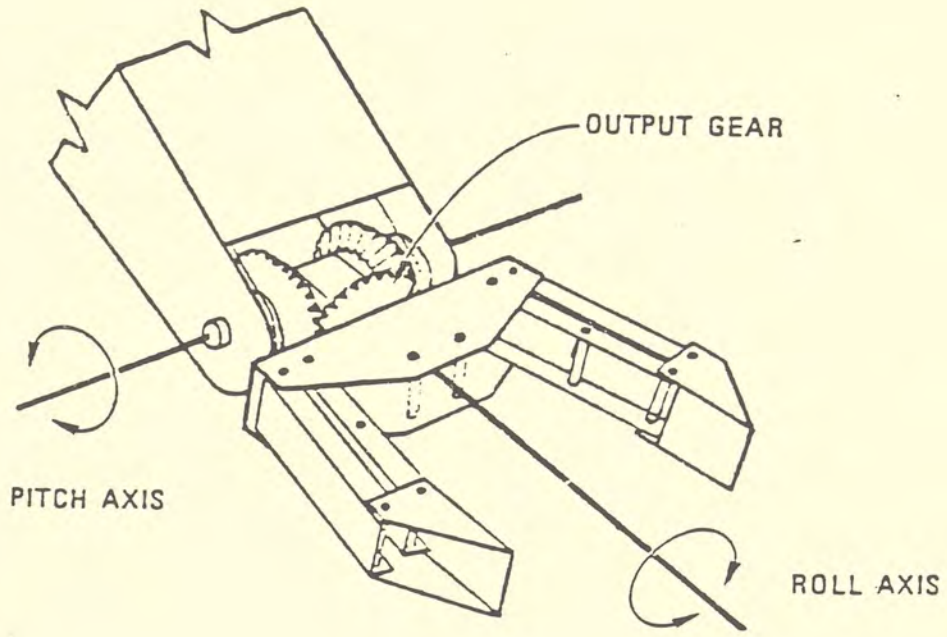


Figure 3. The Wrist Joint (Microbot Inc., 1981)

status results in a pattern of currents in the motor windings. By changing the patterns of currents, a rotating magnetic field is obtained inside the motor. This causes the motor to rotate in small increments or steps.

In the MiniMover-5, each of the four coils in the motor is individually controlled from the computer. The simplified drive circuit is shown in Figure 4.

The relationship between the torque output of the stepper motors and stepper motor speed and the load can be seen in Figure 5.

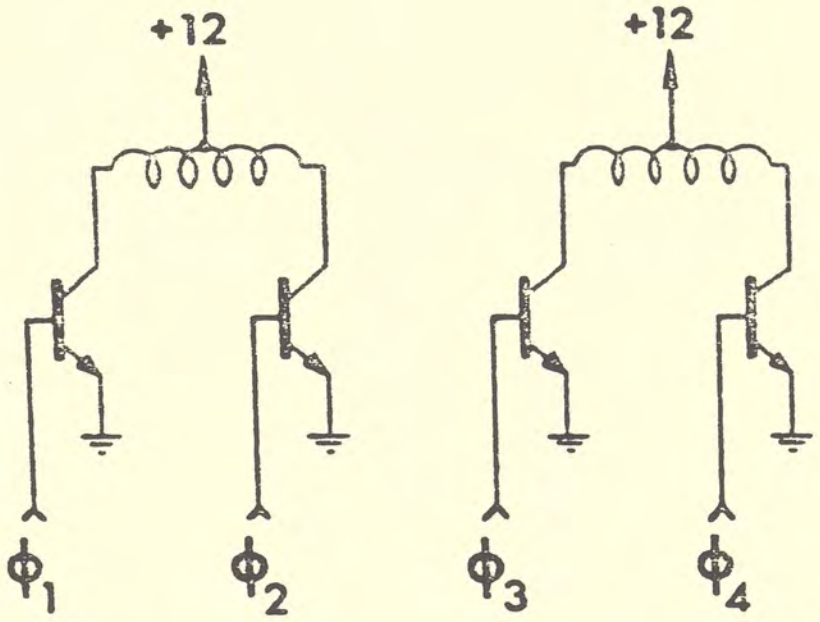


Figure 4. Simplified Stepper Motor Drive Circuit (Microbot Inc., 1981)

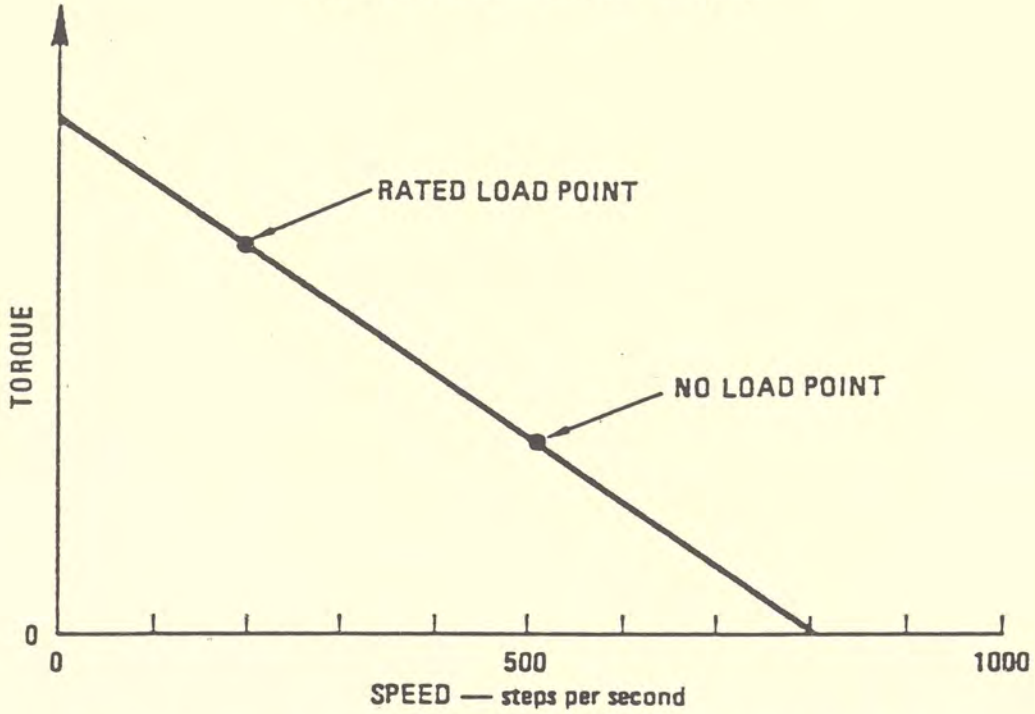


Figure 5. Stepper Motor Speed-Load Tradeoff (Microbot Inc., 1981)

CHAPTER III

THE INTERFACING CARD AND THE ARMBASIC LANGUAGE

As mentioned before, the MiniMover-5 robot can be controlled by a Apple Plus PC. Communication is made possible by the firmware card developed by Microbot, Inc. In order for the MiniMover-5 robot to communicate with a Apple Plus computer, the firmware card must be inserted in one of the Apple Plus computer's peripheral slots. Once the card is inserted the computer will have the complete control of the arm from both Integer Basic and Applesoft (floating point Basic).

Within the ROM (read-only memory), located on the firmware card, resides the MiniMover-5 robot language called ArmBasic. ArmBasic consists of six commands which allow complete control of the manipulator. These commands are:

1- @STEP - Moves the manipulator.

2- @CLOSE - Closes the hand.

3- @SET - Allows the manipulator to move under manual control.

4- @RESET - Zeros the arm position and motor currents.

5- @READ - Returns the current motor position of the manipulator motors.

6- @ARM - Selects the port number which ArmBasic will use.

All of the above commands can be executed directly by the keyboard with the exception of the @READ command. All commands can be executed in programs written in either Integer Basic or Applesoft by using the PRINT statement.

The @STEP command causes the six stepper motors to move simultaneously. The sign and the magnitude of each of the six integer variables used in the syntax defines the direction and the number of steps that each motor will move. The @CLOSE command causes the hand to close until the grip switch indicates that gripping force has built up. This occurs either when the fingers touch each other or an object. The command @SET puts the manipulator into "manual" mode. In manual mode the motion of each joint is controlled from the keyboard (see Figure 6). The @READ command allows the contents of the six internal position registers, which keep track of the number of the steps sent to each joint, to be read into variables within the

registers, which keep track of the number of the steps sent to each joint, to be read into variables within the program only. The @RESET command zeros the contents of the internal position registers. Finally the @ARM command permits the ArmBasic to control one or two robot arms.

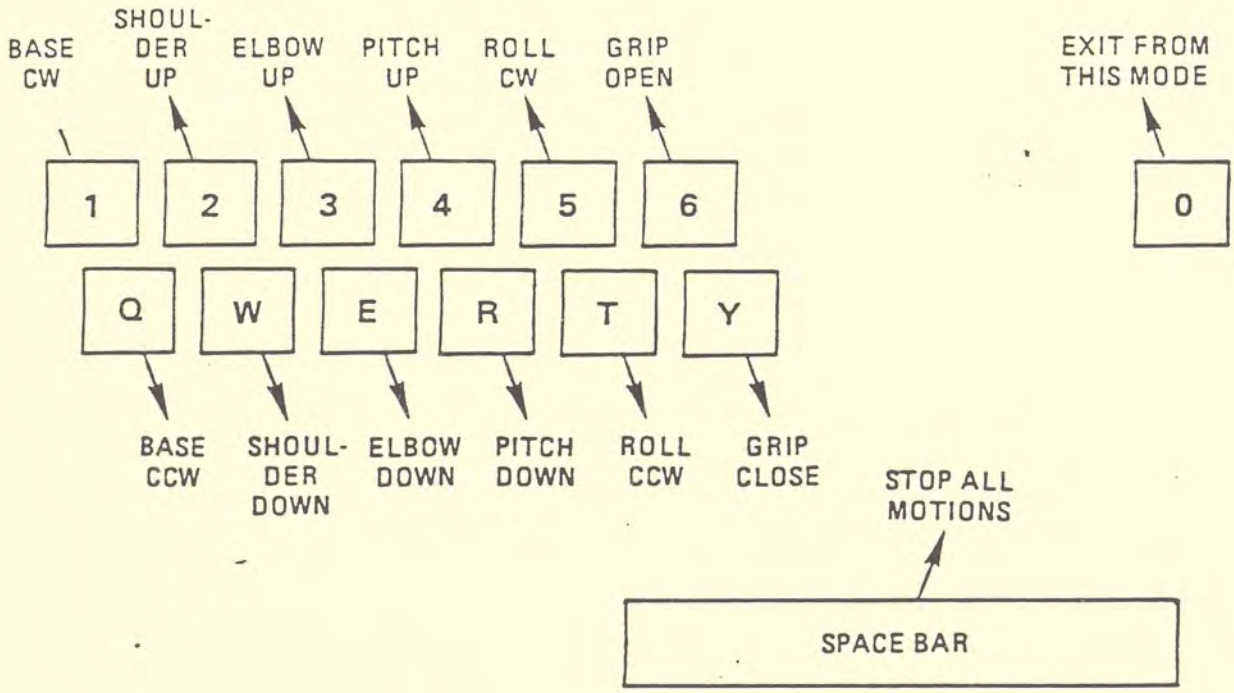


Figure 6. Use of Keyboard for Manual Control (Microbot, Inc. 1981)

CHAPTER IV

THE SOFTWARE

The ArmBasic programming language is categorized as joint-space. This means that the programmer must work on the robot joint-space in order to move the robot arm to a desired location and orientation. Let's consider the final robot location and orientation as point B. Where B is defined in terms of x, y, z, pitch and roll. The programmer must convert the location B from the cartesian coordinate system to the joint coordinate system. In the joint coordinate system, the location B would be defined in terms of angles in degrees from the "home" for each joint to be moved.

Since in a revolute system a straight line is made up of tiny arcs, tasks that require a straight line path also require the user to define the arcs and the respective angular displacements for each robot joint. This type of programming is very tedious and therefore the programmer is likely to make errors. The software developed reduces programming errors by providing a more

more friendly interaction between robot and man. The software will also allow the programmer to work within the cartesian coordinate system. This allows the programmer to simply define the final location and orientation of the robot.

This software consists in a series of English-like commands and an edit mode. As the user types the desired command, the software will request the specific data. The commands available and their functions are as follows:

1- MOVE - This command moves the robot to the final location and orientation without defining the path traveled. The variables to be entered are x, y, z, pitch, and roll.

2- MOVES - This command does the same as the MOVE command but the path traveled by the robot arm is defined as a straight line. In addition, the initial hand orientation is kept constant until the robot arm reaches the end-point. Then, the hand assumes the user specified hand orientation. The variables to be input are x, y, z, pitch, and roll.

3- APPROACH - This command moves the robot arm to the given offset distance from the final location and orientation along its hand z-axis. The command is not

concerned with the path travelled. The inputs required by the command are x, y, z, pitch, roll, and offset distance.

4- APPROACHS - This command performs the same as the command APPROACH but it travels in a straight line path. The variables to be entered are the same as the command APPROACH.

5- DEPART - This command moves the robot arm to a given offset distance from its current position and orientation. The command does not specifically define the path travelled. Therefore, only the initial and final x and y hand coordinates are equal. However, the hand orientation is kept constant during the trajectory. The only input required from the user is offset distance value.

6- DEPARTS - This command performs the same action as the command DEPART but the travelling path is specified as a straight line. Therefore, x and y hand coordinates are kept constant during the trajectory, as well as the the hand orientation. The variable to be input is the offset distance.

7- OPEN - This command opens the robot hand to a specified size. The command requests the user to input the hand size

8- CLOSE - This command closes the robot hand to specified size. The only variable to be entered by the user is hand size.

Each of the commands above is written in the Applesoft language.

Due to the computer limitation, commands which require the generation of points in order to describe a straight line path, are very slow in execution.

CHAPTER V

THE ROBOT COMMANDS

This chapter will describe the function, and the mathematical and programming approaches of the MOVE, MOVES, APPROACH, APPROACHS, DEPART, DEPARTS, OPEN, and CLOSE commands.

The command MOVE moves the robot arm from the current to the final point without being concerned with the traveling path. On the other hand, the command MOVES is used to perform the straight line traveling path. This command calculates all the points that will describe the path, and it uses the command MOVE, as a subroutine, to take the robot arm through each of the calculated points.

The commands APPROACH, APPROACHS, DEPART, and DEPARTS are subsets of the two main commands, MOVE and MOVES. APPROACH, APPROACHS, DEPART, and DEPARTS calculate the offset location from the end point by maintaining the same end-point hand orientation. Then, the commands MOVE

or MOVES is used, as a subroutine, to perform the robot arm motion.

The MOVE Command

The command MOVE moves the robot arm from the current to the final position and orientation without concern for the path traveled by the robot arm. This command converts the desired robot arm end point from the Cartesian coordinate system to the Joint coordinate system through the robot kinematic equations. The Joint coordinate system expresses the angular displacement of each robot joint (base, shoulder, elbow and wrist). This processing, called the Backward Solution and is described below. Applications of the MOVE and similar commands are primarily for pick-and-place tasks.

The equations (1 to 20), used in the command MOVE, are adapted from Microbot, Inc. (1981) (MiniMover-5 manual). Consider figure 7 as the MiniMover-5 robot kinematics model and point B as the final location and orientation. The base joint angular displacement (A1) can be defined as shown below.

$$A1 = \arctan (Y/X) \qquad (1)$$

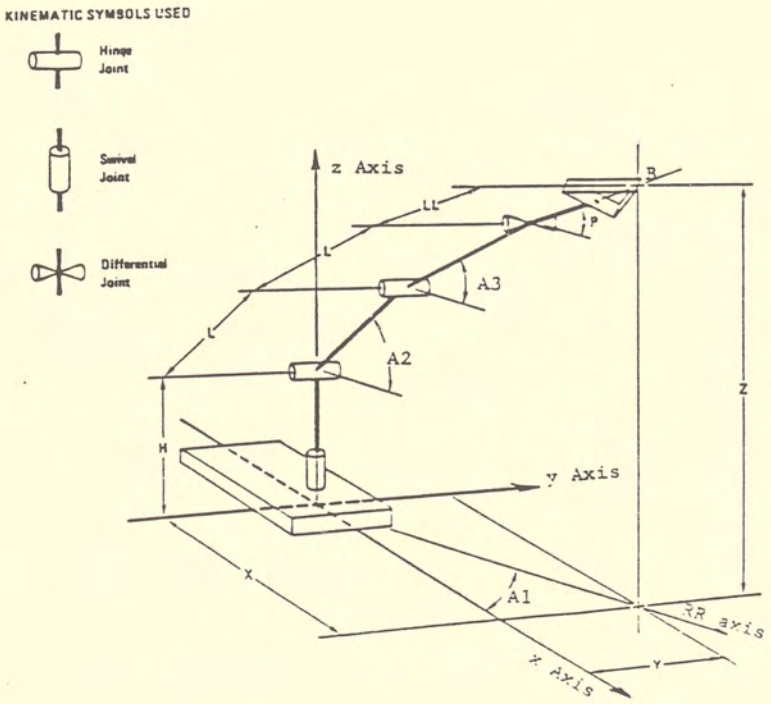


Figure 7. Kinematics Model of the MiniMover-5 (Microbot Inc., 1981)

Pitch and roll angles define the hand orientation (see figure 8) and they are expressed as:

$$P = (A5 + A4) / 2 \quad (2)$$

$$R = (A5 - A4) / 2 \quad (3)$$

where: P - the pitch angle

R - the roll angle

A4 - the right wrist angle

A5 - the left wrist angle

A very special and useful case occurs when the hand is pointing straight down (P = -90°). In this case the roll can be measured with respect to the arm or to the Cartesian frame. The hand orientation can be fixed along the y coordinate axis by subtracting A1 from the roll angle, while the hand is pointing straight down.

Consider the hand being oriented 90° with respect the arm as shown in figure 9. In order to keep the hand fixed with respect to the y-axis, we must have

$$R' = R - A1 \qquad (4)$$

where: R - the roll angle measured with respect the arm.

R'- the roll angle measured with respect the cartesian frame.

To differentiate the normal roll, R, from the cartesian roll, R', a special variable, R1, is introduced. If R1 equals 1 the roll angle is measured with respect to the y-axis. Otherwise if R1 equals 0 the roll angle is

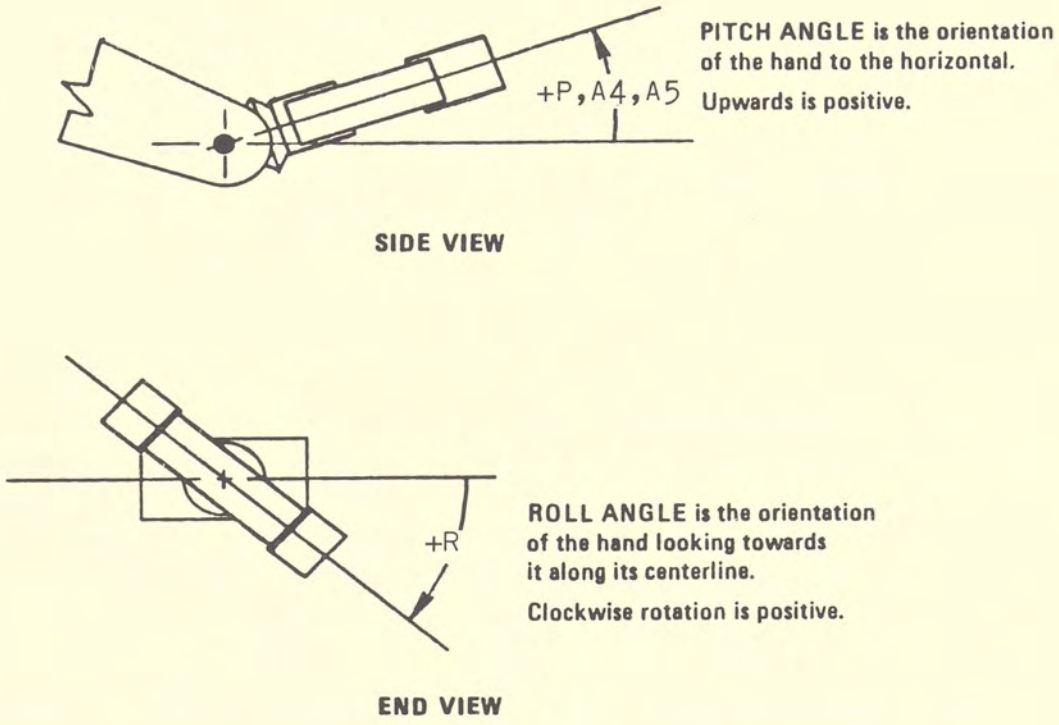


Figure 8. Different Hand Orientation (Microbot Inc., 1981)

measured with respect to the arm. Therefore, the equation 4 can be expressed as:

$$R' = R - (A1 * R1) \tag{5}$$

By solving equations 2 and 3, and substituting for R in equation 5, we have

$$A4 = P + R' + (A1 * R1) \tag{6}$$

$$A5 = P - R' - (A1 * R1) \tag{7}$$

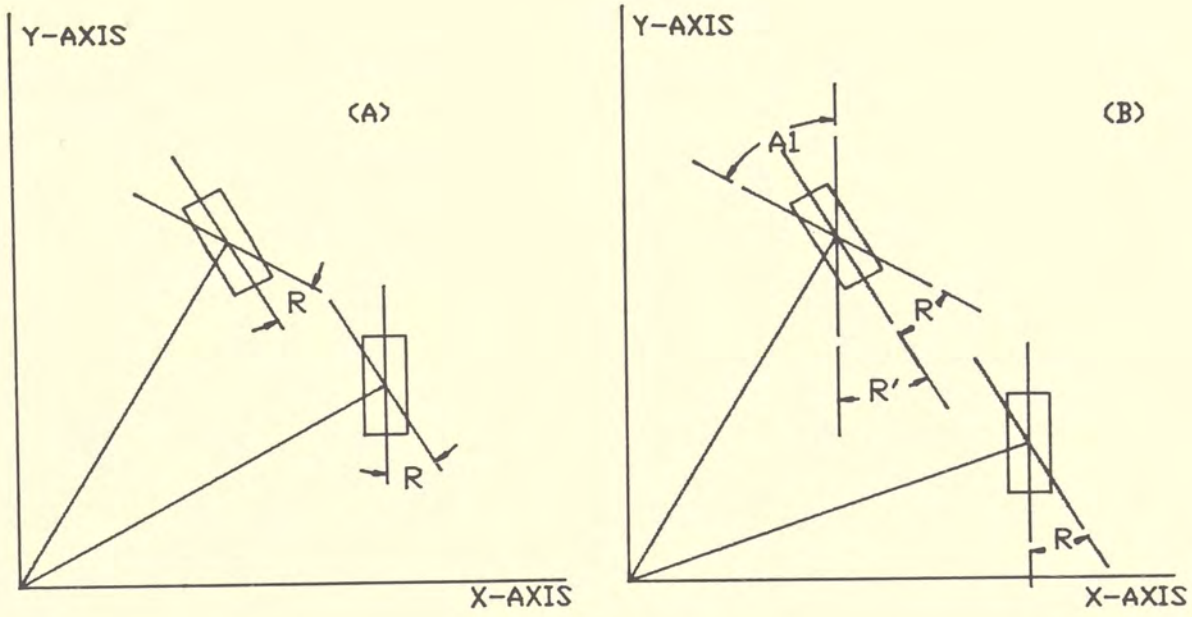


Figure 9. Roll in Arm Frame (a) & Cartesian Frame (b) (Microbot Inc., 1981)

In addition to the angular displacements of the base and wrist joint are the angular displacements of the shoulder and elbow joints. Figure 10 shows the side view of MiniMover-5 kinematics model. The vertical distance (Z), from the table top to the final position, can be expressed by:

$$Z = H + L * \sin(A2) + L * \sin(A3) + LL * \sin(P) \quad (8)$$

where:
- H - height of the shoulder
- L - length of the link
- LL - length of the hand
- A2 - shoulder angle
- A3 - elbow angle
- P - pitch angle
- Z - z-coordinate

The horizontal distance (RR) is the projection of the robot arm on the xy plane for a given position, as shown in Figure 11.

$$RR = \sqrt{(X^2 + Y^2)} \quad (9)$$

Shoulder and elbow angles are defined by using a new coordinate system Z-R with its origin located on the shoulder pivot point as shown in the Figure 12. The variable R0, shoulder to wrist distance, can be expressed as: (See Figure 10)

$$R0 = RR - LL * \cos(P) \quad (10)$$

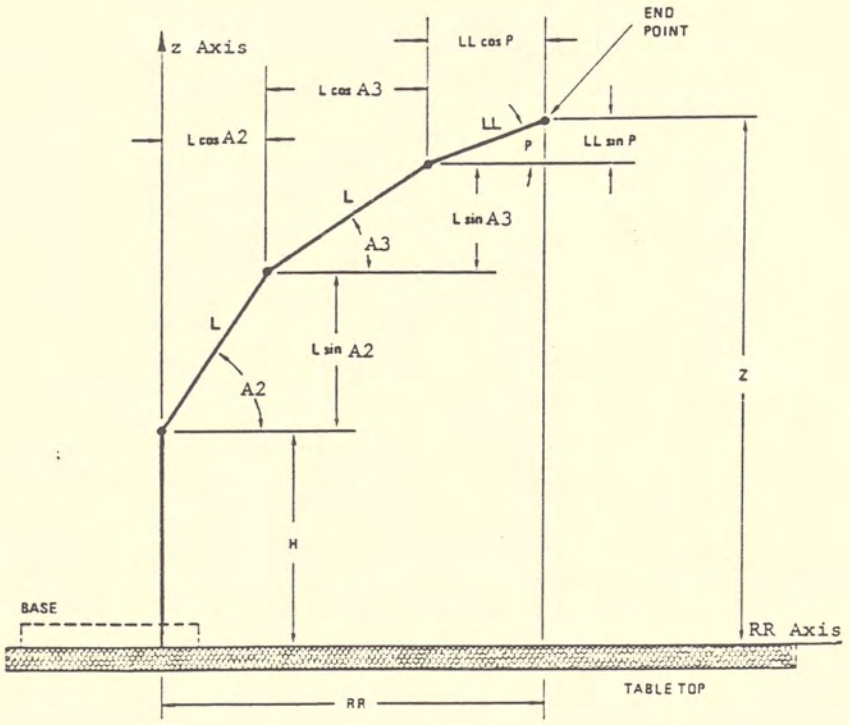


Figure 10. Side View of Kinematics Model (Microbot Inc., 1981)

The height of the wrist above the shoulder (Z0) is defined as: (See Figure 10)

$$Z0 = Z - LL * \sin(P) - H \qquad (11)$$

The variables A6, A7 and A8 are auxiliary angles. From Figure 12 the shoulder angle (A2) can be expressed as:

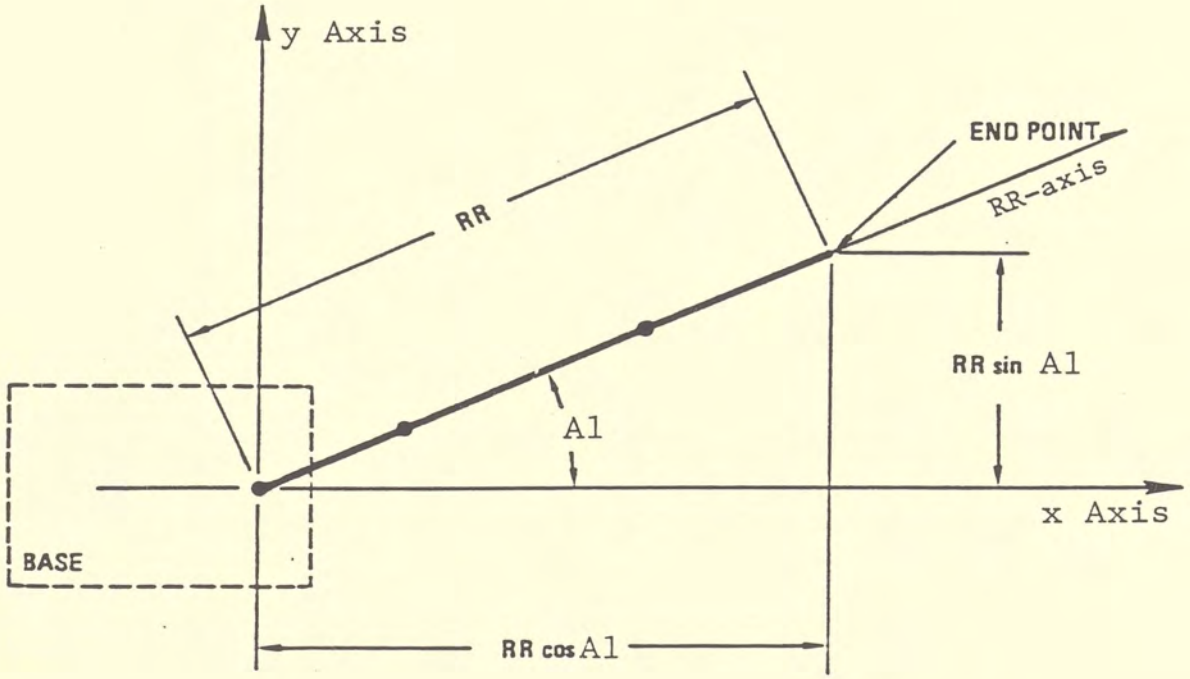


Figure 11. Top View of Kinematics Model
(Microbot Inc., 1981)

$$A2 = A6 + A7 \quad (12)$$

$$A7 = \arctan (Z0/R0) \quad (13)$$

The new shoulder-elbow-wrist triangle configuration is shown in Figure 13. This configuration is the result of pivoting the triangle about the shoulder pivoting point with angle A7.

$$b = \sqrt{(Z0^2 + R0^2)} \; / \; 2 \qquad (14)$$

$$h = \sqrt{(L^2 + b^2)} \qquad (15)$$

The auxiliary angle A6 can be expressed as:

$$A6 = \arctan (h/b) \qquad (16)$$

Substituting equations 14 and 15 in equation 16, we have:

$$A6 = \arctan ((4 (L^2) / (R0^2 + Z0^2) - 1)) \qquad (17)$$

The elbow angle can be expressed as:

$$A2 + A8 + A3 = 180° \qquad (18)$$

Since the sum of internal angles of a triangle is equal to 180°, we have:

$$A6 + A6' + A8 = 180° \qquad (19)$$

where: $A6' = A6$

Substituting equations 12 and 19 into equation 18 we have:

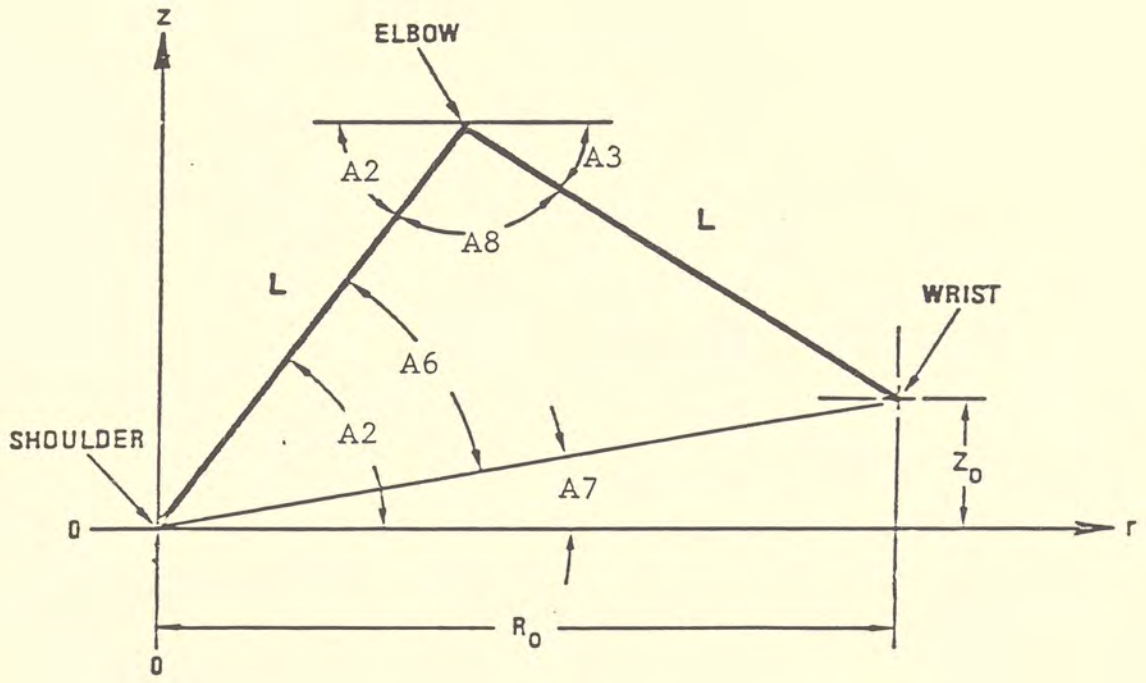


Figure 12. Shoulder-Elbow-Wrist Triangle
(Microbot Inc., 1981)

$$A3 = A6 - A7 \tag{20}$$

Angle A3 was previously defined as the angle above the horizontal as shown in figure 7; therefore, the sign of angle A3 must be changed.

$$A3 = A7 - A6 \tag{21}$$

Table 1 (Appendix A) shows the MiniMover-5 robot range of performance. To ensure that the execution of the

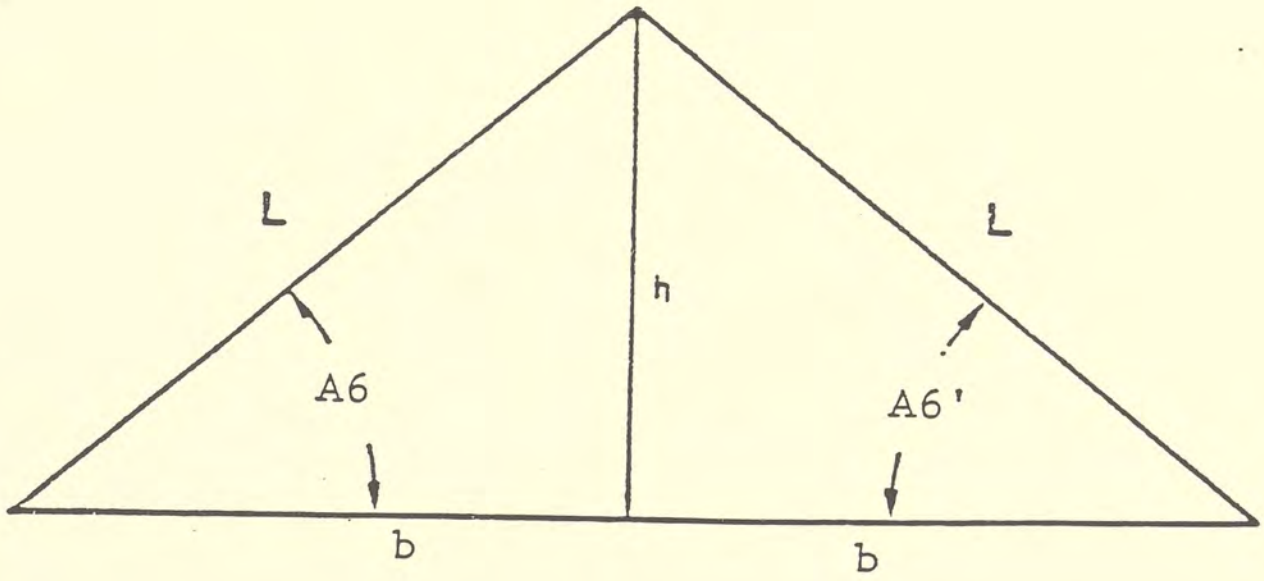


Figure 13. Simplified Triangle
(Microbot Inc., 1981)

MOVE command program does not violate the MiniMover-5 range of performance, several restrictions are imposed. These restrictions are defined in the following paragraphs.

The closest distance to the robot body that the end effector can be positioned is determined by the configuration of the robot. It is 2.25 inches on the

RR-axis as given in Robotics Reference and Application Manual of MiniMover 5. Therefore,

$$RR >= 2.25 \text{ in.} \quad \text{(See Figure 10)} \tag{22}$$
$$X > 0 \quad \text{(See Figure 1)}$$

Table 1 (Appendix A) shows the working range of the pitch and roll angles. Since the they are defined by the left and right wrist angles in equations 5 and 6, we have

$$-270° <= A4 <= 270° \tag{23}$$
$$-270° <= A5 <= 270° \tag{24}$$

The RR value, by itself, does not guarantee the avoidance of a wrist-body collision. For example, if the end point of the robot arm is defined as X = 2.25 in., Y = 0 in., Z = 0 in., P = 0 deg., and R = 0 deg. The conditions of equation 22 would be satisfied but this would not prevent a wrist-body collision. See Figure 14. By limiting the range of RO values, the wrist-body collisions are avoided (See Figure 12).

$$RO >= 2.25 \text{ in} \tag{25}$$

The limits of the shoulder and elbow angles are functions of angles A6 and A7. From the equation 16, we can confirm that A6 must be:

$$A6 >= 0$$

Since shoulder and elbow angles are governed by the equations 12 and 21. The shoulder range (see Table 1 - Appendix A) is

$$-35° <= A2 <= 144°$$

and the elbow range (see Table 1 - Appendix A) is

$$0° <= (A2 - A3) <= 149° \quad (26)$$

The pitch range is a function of the length of the arm. The retraction or extension of the arm will affect the pitch range. Consider the hand coordinate system shown in Figure 15. The pitch angle is measured with respect to the initial z-axis orientation of the hand, which is aligned with the forearm. An arbitrarily

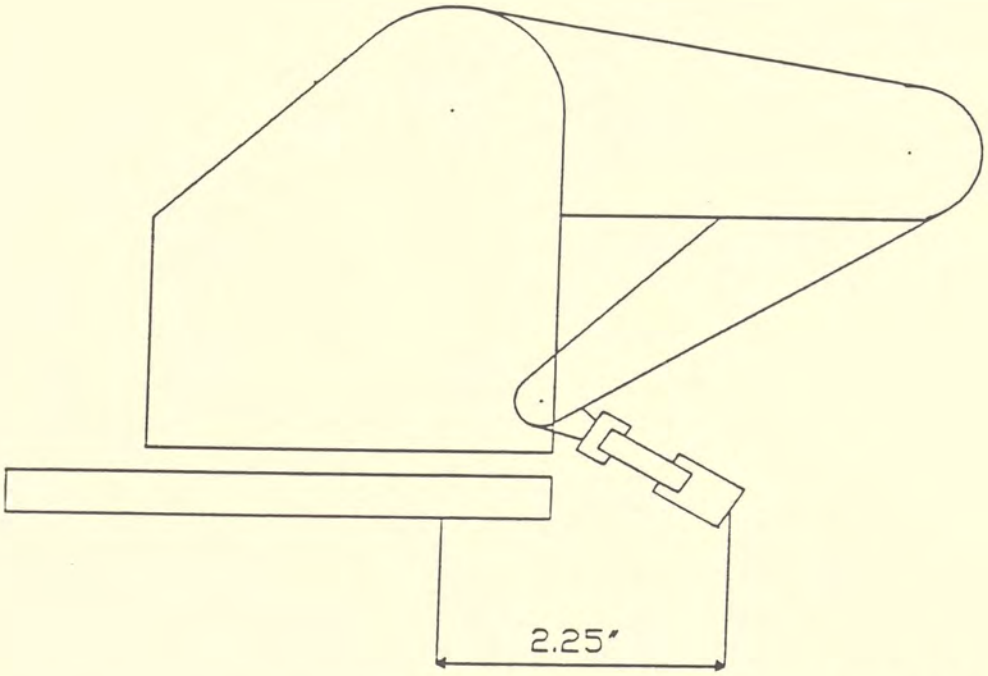


Figure 14. The Improper Robot End-Effector Position

end-point is given in Figure 16 to show how to determine the pitch range. The pitch range is defined when the hand makes an angle of ± 90 degrees to the forearm. Since the retraction or extension of the arm changes the pitch range, we can define the pitch range as:

$$(90^\circ - A3) <= P <= (90^\circ + A3) \qquad (27)$$

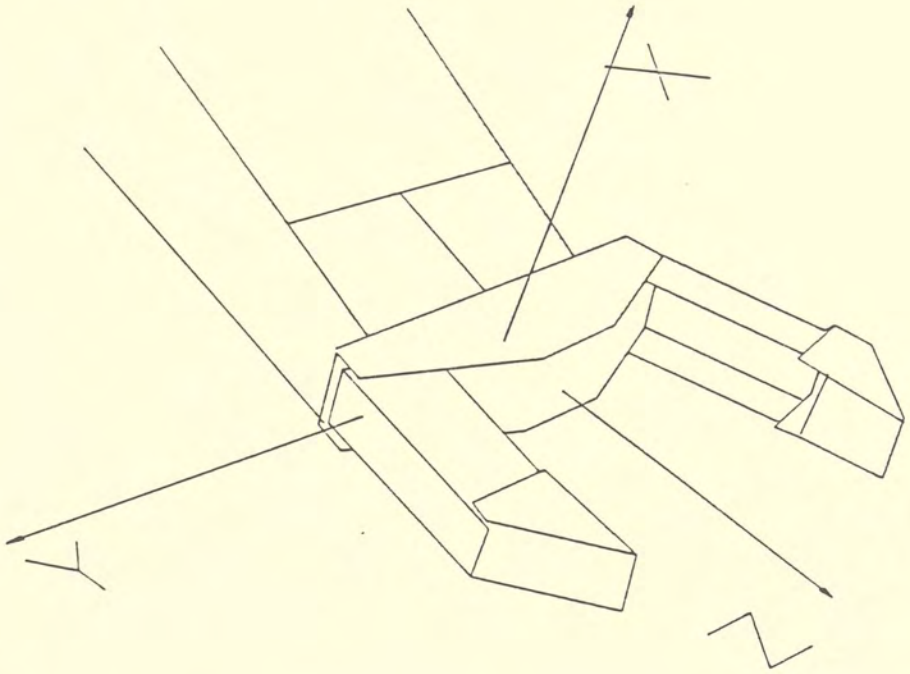


Figure 15. The Hand Coordinate System

The roll angle is measured with respect to the "home position". The roll range is defined as

$$-180° <= R <= 180°$$

The command MOVE program flowchart is shown in the Appendix B.

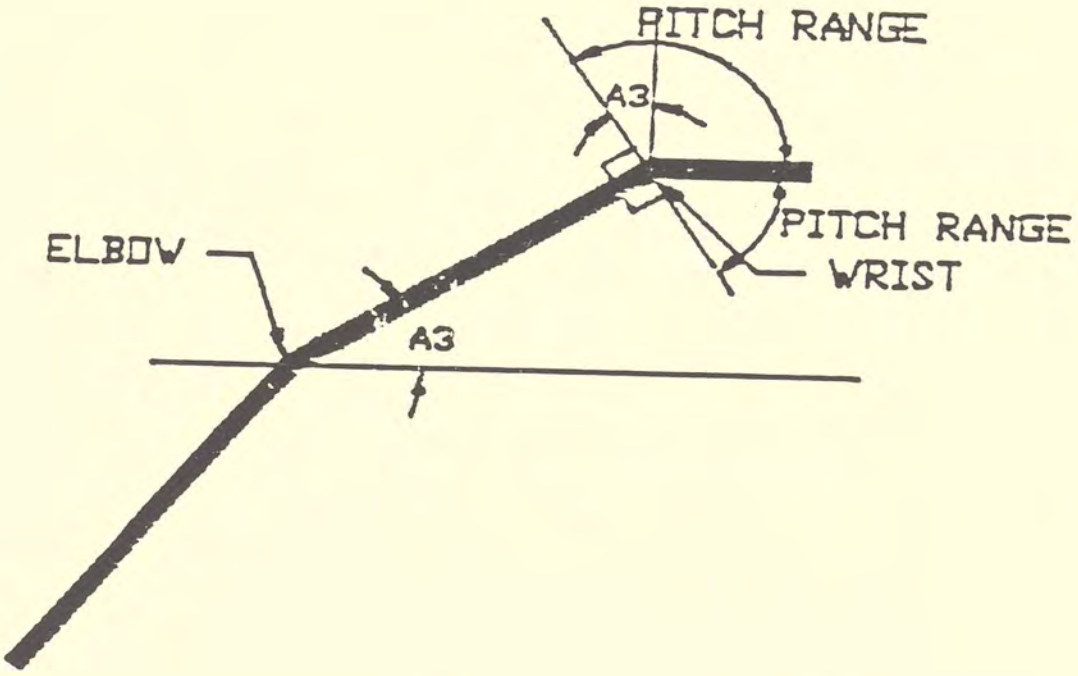


Figure 16. The Pitch Working Angle Range

The MOVES Command

The command MOVES performs identically to the command MOVE, except the path followed by the arm is a straight line.

Reducing the robot travelling distance and inserting a pin in a hole are two of many tasks that can be accomplished by the MOVES path control feature. The MiniMover 5 can incorporate the path control feature through the use of the command MOVES.

A path control machine is no more than a point-to-point motion control machine with a special software. A motion control machine is only concerned with the initial and final positions. The software simulates a straight line path by calculating many tiny point-to-point movements (PTP). The equations that transform PTP machine to a path control machine are described below.

Consider an arc as being the trajectory described in a PTP robot movement. See Figure 7 and 17. An arc can approximate a straight line by limiting the variable T. Therefore:

$$RT = RR - T \tag{28}$$

where: T - tolerance between the desired and actual path
RR - radius of the robot arm

The number of PTP movements is a function of the segment length and the specified tolerance. See Figure 18.

$$\cos (A9 / 2) = RT / RR \tag{29}$$

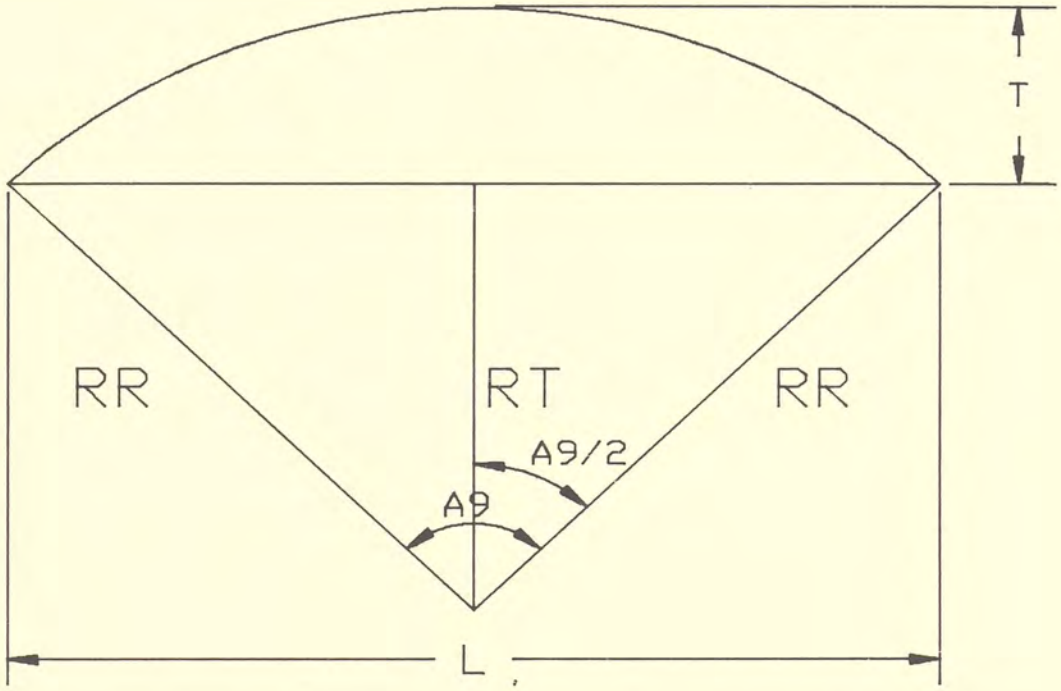


Figure 17. Trajectory of a PTP robot

Using equation 28 and substituting into equation 29 we have:

$$A9 = 2 * \arccos ((RR - T) / RR) \tag{30}$$

The segment length can be calculated by applying the cosine law.

$$L^2 = RR^2 + RR^2 - 2\ RR^2 * \cos(A9)$$

$$L = RR * \sqrt{(2 * (1 - \cos(A9)))} \tag{31}$$

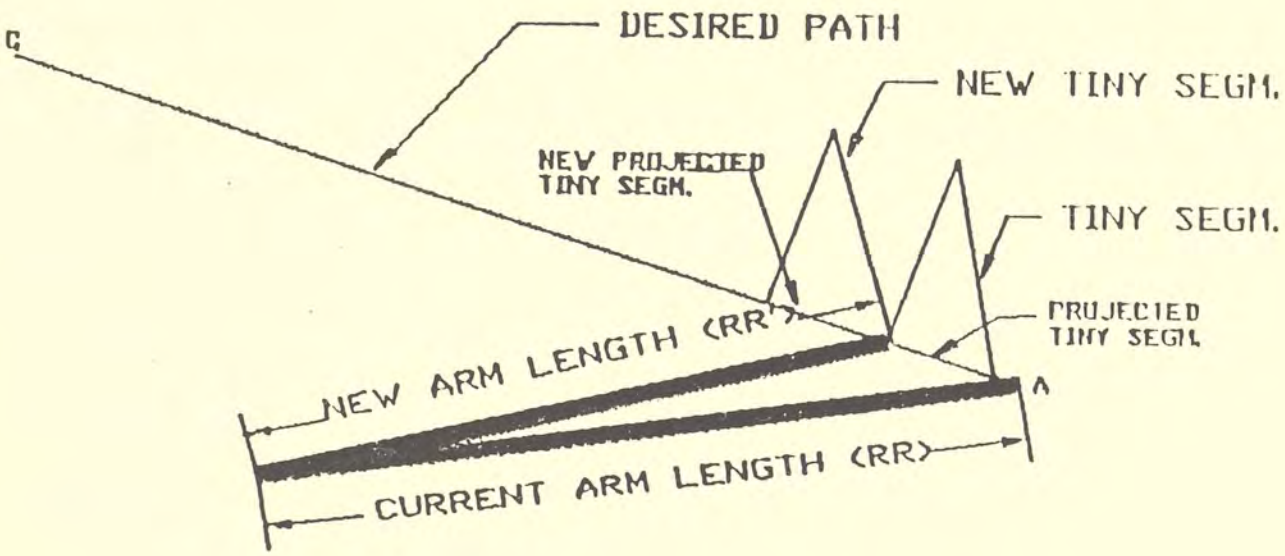


Figure 18. The Straight Line Path

For the purposes of this research, when a straight line connects two points (one of the points must have the coordinate x, y and z different than zero), it is called a spatial straight line. The approach adopted to calculate the PTP segments of a spatial straight line are: First, decompose it on all three plane (XY, XZ, and YZ). The now decomposed components of the spatial straight line are called plane straight lines. Second, determine the PTP segments of the plane straight line of the XY-plane. Third, select either an x or y coordinate of each

calculated segment on the XY-plane and calculate the z coordinate of each segment using the equation of a line on XZ or YZ plane. This process will become more clear further along in the chapter.

Since the length of the segments are not constant (they change according to the value RR) their coordinates must be defined individually. The set of equations developed in the next paragraphs are used to fulfill this need.

Given that the robot has been assigned to perform a plane straight line motion from A to C. (See Figure 19.)

$$L1 = \sqrt{((XA - XC)^2 + (YA - YC)^2)} \tag{32}$$

$$RR1 = \sqrt{(XC^2 + YC^2)} \tag{33}$$

where: L1 - length of the straight line

RR1 - final robot radius

Apply the cosine law to define the angle A10:

$$RR1^2 = RR^2 + L1^2 - 2 * RR * L1 * \cos(A10)$$

$$A10 = \arccos((RR^2 + L1^2 - RR1^2) / (2 * RR * L1)) \tag{34}$$

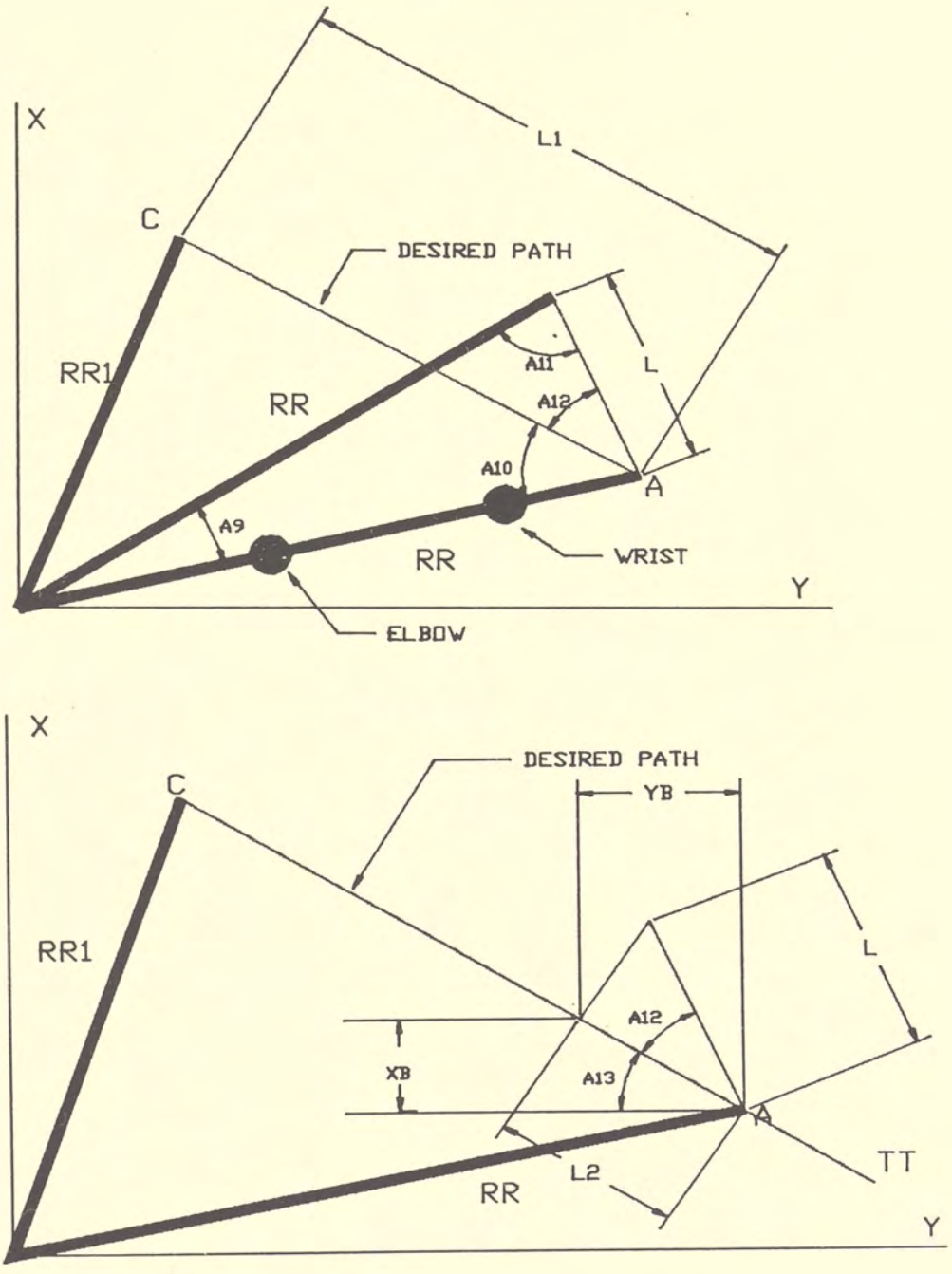


Figure 19. Straight Line Motion on the Plane

Since the sum of the internal angles of the triangle OAB is equal 180, we can define the angle A11 as:

$$A11 = (180° - A9) / 2 \quad (35)$$

The angle A12 can be defined as:

$$A12 = A11 - A10 \quad (36)$$

Consider the desired path (straight line) being on the TT-axis. (See Figure 19).

$$L2 = L * \cos (A12) \quad (37)$$

The TT-axis makes an angle A13 with the y-axis. The components of the segment L2 in x and y axis can be defined as:

$$YB = L2 * \cos (A13) \quad (38)$$

$$XB = L2 * \sin (A13) \quad (39)$$

Using equations 38 and 39 we can define the absolute final position of the PTP movement as being:

$$X = XA + XB \tag{40}$$

$$Y = YA + YB \tag{41}$$

The equations 40 and 41 define the x and y cartesian coordinates of the xy-plane.

The same concept is valid to calculate the PTP segment coordinates of a plane straight line trajectory on the XZ or YZ plane. The equations that calculate the PTP segments on the XZ-plane are:

$$RT = RR - T \tag{42}$$

$$\cos(A9 / 2) = RT/RR \tag{43}$$

$$A9 = 2 * \arccos((RR - T) / RR) \tag{44}$$

$$L = RR * \sqrt{(2 * (1 - \cos(A9))} \tag{45}$$

$$L1 = \sqrt{((ZA - ZC)^2 + (XA - XC)^2)} \tag{46}$$

$$RR1 = \sqrt{(ZC^2 + XC^2)} \tag{47}$$

$$A10 = \arccos((RR^2 + L1^2 - RR1^2) / (2 * RR * L1)) \tag{48}$$

$$A11 = (180 - A9)/2 \tag{49}$$

$$A12 = A11 - A10 \tag{50}$$

$$L = L * \cos(A12) \tag{51}$$

$$XB = L2 * \cos(A13) \tag{52}$$

$$ZB = L2 * \sin(A13) \tag{53}$$

$$X = XA + XB \tag{54}$$

$$Z = ZA + ZB \tag{55}$$

$$ZB = L2 * \sin(A13) \quad (53)$$

$$X = XA + XB \quad (54)$$

$$Z = ZA + ZB \quad (55)$$

The equations that calculate the tiny PTP segments on the YZ-plane are:

$$RT = RR - T \quad (56)$$

$$\cos(A9 / 2) = RT/RR \quad (57)$$

$$A9 = 2 * \arccos((RR - T) / RR) \quad (58)$$

$$L = RR * \sqrt{(2\ (1 - \cos(A9))} \quad (59)$$

$$L1 = \sqrt{((XA - XC)^2 + (ZA - ZC)^2)} \quad (60)$$

$$RR1 = \sqrt{(ZC^2 + XC^2)} \quad (61)$$

$$A10 = \arccos((RR^2 + L1^2 - RR1^2) / (2 * RR * L1)) \quad (62)$$

$$A11 = (180 - A9) / 2 \quad (63)$$

$$A12 = A11 - A10 \quad (64)$$

$$L = L * \cos(A12) \quad (65)$$

$$ZB = L2 * \cos(A13) \quad (66)$$

$$XB = L2 * \sin(A13) \quad (67)$$

$$X = XA + XB \quad (68)$$

$$Z = ZA + ZB \quad (69)$$

$$X = XA + XB \quad (68)$$

$$Z = ZA + ZB \quad (69)$$

The sets of equations developed above are only valid for describing a straight line path on a plane. Most probably, the path control requested by many of the robot tasks is a spatial straight line. Therefore, the sets of equations developed above must be combined in order to calculate the coordinates of a minute spatial segment.

The connection between the set of equations of XZ-plane is made by the equations below.

$$Z = ZA + SC * (X - XA) \tag{70}$$

$$SC = (ZC - ZA) / (XC - XA) \tag{71}$$

where: SC - slope of the line

The x or y value is obtained from the calculation of the first tiny segment coordinate in the XY-plane, in equation 70. Given the initial and final location, the procedure is to use the x or y value to calculate the corresponding z coordinate of the first tiny segment.

The connection between the set of equations of YZ-plane is made by the equations below:

$$Z = ZA + SC * (Y - YA) \tag{72}$$

$$SC = (ZC - ZA) / (YC - YA) \quad (73)$$

The procedure that calculates the z coordinates of each spatial tiny segment is the same as the one explained above.

The flowchart of the program and its explanation is given in the following paragraphs. The command MOVES program flowchart can be seen in Appendix C. The description of the program flowchart is given below.

When the program is executed, it is initiated by requesting the final location and orientation data . Then, the program reads the position of each motor to obtain the current location and orientation of the MiniMover5 arm . Once the initial and final locations and corresponding orientation are known, the program determines the nature of the straight line; plane line or spatial line. If it is a spatial straight line, the next step is to calculate of all the necessary segment coordinates that will define the decomposed straight line path on the XY-plane. Otherwise, the program branches to the specific block of codes which calculates all the segment coordinates that describe the straight line path on the respective plane.

At this stage, in order to calculate the segment coordinates, the program calculates the coordinate of the

first segment and compares it to the coordinate of the final location. If the last segment coordinate does not reach the final location, an additional segment coordinate is calculated and compared. This process continues until a segment coordinate reaches or passes the known final coordinate. If the last segment coordinate passes the final coordinate, the calculated coordinate is made equal to the given final coordinate.

If the straight line is either a plane line on XY-plane or a spatial line, the program will branch to the block of codes that will calculate the segment coordinates on the XY-plane. Once the coordinates are defined, the nature of the straight line is rechecked. For a plane straight line, the program will write the coordinates to a file called "Data". This file will be used as the input of the command MOVE program. The MiniMover 5 can then perform the desired straight line motion . For spatial straight lines, the program will calculate the z-coordinate of each segment. The z-coordinates are calculated by substituting either an x or y coordinate into equations 70 or 72 respectively. Therefore, a tiny segment on the XY-plane might not be a tiny segment on XZ or YZ plane. One method, which guarantees that the variable T (equation 28) will not be larger than the adopted value 0.01 inches (maximum

MiniMover-5 resolution), is to treat each minute segment on either XZ or YZ plane as a desired straight line path. By adopting this method, a set of segment coordinates for each minute straight line is defined. Once all the segments coordinates are defined, the command MOVES program also rewrites them to a file called "DATA". Finally, the command MOVE is executed with its input coordinates available from the file "DATA". The equations used for the command MOVE are used for each of the x, y and z coordinates defined by the equations of command MOVES.

The APPROACH / APPROACHS Command

Consider a packaging operation in a manufacturing facility where the product comes in a conveyor, and an operator places it into a box. Generally, this operation can be performed by a robot. If a robot were used, the robot operation cycle would be made up of the following motions:

1. Move to the conveyor downloading point with a set distance above the product
2. Move to the actual position of the product
3. Grasp the product
4. Lift the product to a given height

5. Move the product above the box location
6. Move the product to the unloading point
7. Unload the product
8. Move up and out of the box to a given height
9. Repeat the cycle.

In order to program this automatic packaging task, the programmer must know the end location of each motion given above. The command APPROACH minimizes part of this effort by allowing the user to define only the height for the motions 1 and 5. The APPROACH command can move the robot arm to a given offset distance, along the hand z-axis, from the end point.

The command APPROACH program requests the final location and orientation followed by the offset distance. The program reads the present motor position of each joint and converts them to the current orientation and Cartesian coordinates. Using the given final pitch angle and the offset distance, the program calculates the offset cartesian coordinates and writes them to a file called "Data".

The command APPROACHS behaves similarly to the command APPROACH , but the path to be traveled by the robot is a straight line. Both program are governed by the same equations, and they have the same programming algorithm except that the command APPROACHS program uses

the command MOVES as the subroutine to perform the physical motion instead of the command MOVE. The mathematical approach to calculate the offset coordinates of the end point is described in the next several paragraphs.

Consider the side view of the robot arm as shown in Figure 20. Decomposing the offset segment on the RR-axis and z-axis we have:

$$XY = D * \cos(P) \tag{74}$$

$$Z = D * \sin(P) \tag{75}$$

where: XY - offset segment in RR-axis
Z - offset segment in Z-axis
D - offset distance

From the top view of the robot arm. (See Figure 21).

$$A1 = \arctan(XC / YC) \tag{76}$$

where: XC - x coordinate
YC - y coordinate

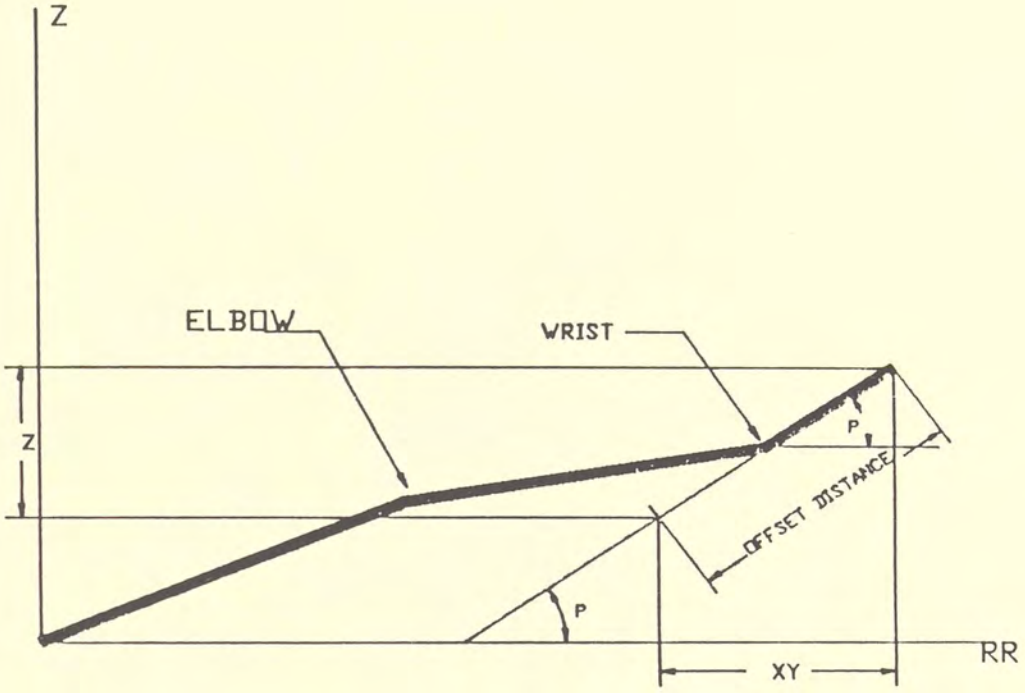


Figure 20. Offset Distance on the ZRR-Plane

Decomposing the offset distance in x and y-axis, we have

$$Y = XY * \cos(A1) \tag{77}$$

$$X = XY * \sin(A1) \tag{78}$$

Therefore, the offset coordinates (XC', YC', and ZC') can be expressed by subtracting equations 75, 77, and 78 from the final positions.

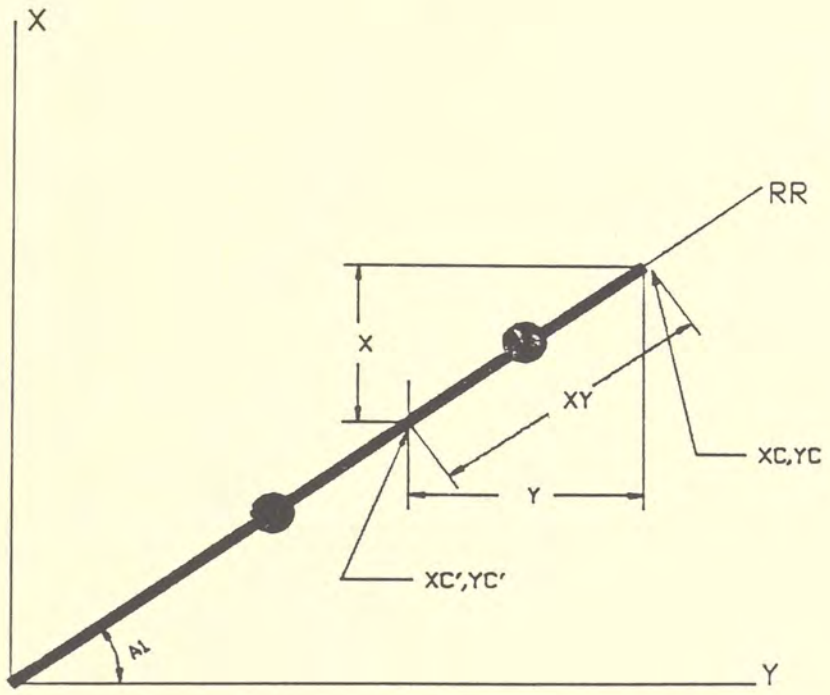


Figure 21. Offset Distance on the XY-Plane

$$XC' = XC - X \tag{79}$$

$$YC' = YC - Y \tag{80}$$

$$ZC' = ZC - Z \tag{81}$$

The commands APPROACH and APPROACHS program flowcharts can be seen in Appendix D.

The DEPART / DEPARTS Command

The commands DEPART and DEPARTS move the robot arm to an offset position (along the hand z-axis) from the

current position. The command DEPARTS moves the robot arm in a straight line path.

When the command DEPART or DEPARTS program is executed it seeks the offset distance. Then, the program reads the current position of the motors to define the current position and to calculate the offset position. The offset position is calculate with the pitch angle constant. Once the cartesian coordinates are calculated, the program writes them to a file called "Data". Depending on the command executed, DEPART OR DEPARTS, the program will uses the command MOVE or MOVES as subroutine to perform the physical robot arm motion. The inputs to these subroutine are stored in the file called "Data".

The equations used by the command DEPART AND DEPARTS, to calculate the offset position, are the same as those used by the commands APPROACH and APPROACHS.

The DEPART and DEPARTS program flowcharts are shown in Appendix E.

The OPEN / CLOSE Command

The commands OPEN and CLOSE control the size of opening of the hand which has the maximum opening size of three inches.

When the command program is executed, it prompts for the final robot hand size. This will be compared to

the robot hand working range. If it is within the correct range, the program will read the current hand motor position, given in step units (309 steps per inch), and convert it to corresponding hand size, also in inches. Using the current and the desired hand size, the program calculates the necessary motor displacement.

The programs OPEN and CLOSE commands' program flowchart are shown in Appendix F.

CHAPTER VI

THE MINIMOVER-5 EDIT AND TEACH MODE

According with the current industrial practice, robot progragmming can be classified in two basic types:

1- Leadthrough methods

2- Textual robot languages

The leadthrough methods, also referred as "Teach-by-showing" methods, consist in moving the robot through the desired motion path in order to record the path into the controller memory. The leadthrough methods can be subdivided into powered and manual methods.

The powered leadthrough method is limited to point-to-point motions. This type of programming is more suitable to robots assigned to perform transfer tasks, machine loading and unloading, and spot welding.

The manual leadthrough methods are more applicable for robots assigned to perform tasks that require continuous path programming such as spray painting and arc welding.

Robot programming with textual languages resembles computer programming. They consist of establishing the logical sequence of a robot program using high-level English like language. However the specific point locations are defined using teach pendant control.

For the objective of this research, the point location, defined by using teach pendant control, is called "Touch Position" or "Touch Input". During the next paragraphs, the procedure used to develop a teach and edit mode for the MiniMover-5 will be described.

The Teach Mode

The MiniMover-5 teach mode simulates a industrial powered leadthrough programming method. However, the MiniMover-5 teach mode does not have the capability of allowing the user to manually guide the the robot arm to the desired position. To lead the arm to the desired location and orientation, the user must use the keyboard as a teach pendant.

The teach mode developed is activated by executing a program called "TEACH". This program initiates by automatically setting up the teach mode for the robot. Once in the teach mode the user can move the robot arm to a desired location by pressing the proper keys on the keyboard (see Figure 7). When the point is reached, the

user exits the teaching stage by entering the number zero (0). The number zero key simulates the record button of a teach pendant. This allows the program to "record" the position of each motor and define the coordinates and orientation, x, y, z, pitch, roll, and hand size, of the just touch position. To save these as a unique position the program requests a name to be given to the position. This name is used as the name of the file which contains the robot arm coordinates and orientation of that particular location.

Once the file is created and the proper information is stored, the program displays a menu with two options, "Continue" or "Exit". The "Continue" option takes the programmer back to the teaching stage, while the "Exit" option displays the main menu.

The Edit Mode

The edit mode developed for the MiniMover-5 robot allows the user to simulate an industrial robot with an off-line textual programming capability. This edit mode is a program written in AppleBasic.

When the computer is powered, a menu is brought up to the screen. By selecting the "Edit a program" option, a program called "Edit" is executed allowing the user to edit a new program. In other hand, the "Reedit a program"

option executes a program called "Edit2", and the programmer is able to change an existing program.

Execution of the program Edit begins by requesting the name of a command. Once the name is supplied, the program checks the syntax of the command, if the name entered does not match with the available commands, Edit displays a error message and requests the user to reenter a new command. If the name of a command is valid, the request for the necessary inputs for the respective command follows. However, the command's inputs can be either a touch or non-touch input; therefore, the user must inform the program when one is entering the command name. This is done by adding the letter T in front of the commands, except for the commands DEPART and DEPARTS. Entering a command with the prefix T, the program prompts a request of the touch position name and/or additional information, depending on the command entered, such as the offset distance and R1. The command names and their inputs are stored in unique vector arrays. If a touch input is given, the program seeks for a file which had the touch position name. Once the file is found, its data is read and attributed to the proper vectors. The editing process continues until the programmer enters the statement "END". When this occurs the Edit prompts for the name of the program to be saved. For this purpose, the programmer is

forced to save the just edited program. The saving process consists in writing the commands' names and their respective inputs to the file of the name that was previously supplied. The first data to be written to this file is a number that indicates the number of lines, which consists of the command and its inputs, that exist in the user program. This number is used by the program Edit2 during the reediting process. With the saving process completed, the main menu is redisplayed.

Choosing the "Reedit a program" option, a program called "Edit2" is executed. It initiates by prompting a request for the name of the program to be reedited. The requested file is loaded and a reedit menu is displayed. Each of the commands and their inputs are also stored in a unique vector array.

The "Insert a line" option allows the user to enter a new command and its inputs in the user program. The "Delete a line" removes a command and its inputs from the user program. The "Replace a line" performs at the same time as the insert and delete a line functions, as well as by itself. The "Save a program" option saves the user program under a given name. Finally the "Exit" option exits from the Edit2 without saving the changes performed in the user program and returns to the main menu.

axis coordinates. The joint-space control task is to convert the desired joint coordinates into the appropriate signal to control the force and torque of the motors. This type of language is hardware dependent. The advantage of this level of control language is the low cost of software development. The disadvantages include the lack of integrated work cell commands required for system control, the absence of any cartesian coordinate values for programmed points, and severe limitations on path and velocity control. Examples of this type of language are found most frequently on the many educational robots such as Armbasic used with the MiniMover-5 educational robot and the Rasp language used with Rhino XR educational robot.

Point-to-point primitive languages are the most common type used on the industrial robots available today. This type of language consists of leading the robot through desired motions and recording the positions and orientations of the robot. The user can create a sequence of steps that can be played back any number of times. The advantages of this type of language are that it is simple to use, easy to debug, and commercial packages are readily available. The disadvantages are little programming

When the "Insert a line" is selected, Edit2 requests the line number where the insertion will occur. Consider the program below.

```
1 MOVE 10,20,10,5,-45,2,0,0
2 APPROACH 20,10,5,5,-45,2,0,2
3 MOVES 20,10,5,5,-45,2,0,0
4 DEPART 20,10,5,5,-45,1,0,3
5 END
```

If the programmer wants to insert the command CLOSE between lines 3 and 4, the programmer will enter 4 as the new line number. When the line number request is supplied, Edit2 moves everything, the commands and their inputs one line down from the line number entered as shown below.

```
1 MOVE 10,20,10,5,-45,2,0,0
2 APPROACH 20,10,5,5,-45,2,0,2
3 MOVES 20,10,5,5,-45,2,0,0
4 CLOSE 20,10,5,5,-45,1,0,0
5 DEPART 20,10,5,5,-45,1,0,3
6 END
```

Then, Edit2 behaves similary to Edit, except that it returns to the reedit menu as soon as the command and its inputs are inserted.

In the case where a program has the "END" statement missing (as shown below), the user has to insert the "END" statement between lines 3 and 4, reinsert the command DEPART, and delete the extra command DEPART.

```
1 APPROACHS 20,10,5,5,-45,2,0,2
2 MOVES 20,10,5,5,-45,2,0,0
3 CLOSE 20,10,5,5,-45,1,0,0
4 DEPART 20,10,5,5,-45,1,0,3
```

When selecting the "Delete a line" option, the line number to be deleted is requested. Once the line number is entered, Edit2 behaves just the opposite to the "Insert a line" option. Edit2 moves all the commands and their inputs from the line below the line to be deleted up one line. The reedit menu is then redisplayed.

If the "Replace a line" option is selected, the program requests the number of the line to be replaced. The values of the vectors which store the command's name and its inputs are overwritten. Then, it returns to the reedit menu.

Selecting the "Save a program" option, Edit2 asks for the name under which the user program will be saved. It verifies the presence of the "END" statement in the the user program. If such a statement is not present, an error

message is displayed. If "END" is present, the just reedited program is saved. The saving process is the same as with Edit. Once the process is completed, the main menu reappears.

In addition to the options mentioned, there are two other options, the "Run a program" and the "Print a program".

The "Run a program" option executes the given user program. This process is performed through the execution of a program called "Edit1". Before Edit1 is executed, the name of the user program to be run is requested, and a file called "Counter" is created. This file contains the name of the user program, the size of the program in terms of line number, and the line to be executed next. When Edit1 is executed, it starts by reading Counter. Then, Edit1 checks if the next line to be executed is the same as the line that contains the "END" statement. If so, the main menu is displayed; otherwise, Edit1 seaches for the command and its inputs on the respective line. Once they are found, the old file called "Counter" is overwritten with basically the same type of information. The difference is the value of the next line of the user program to be executed.

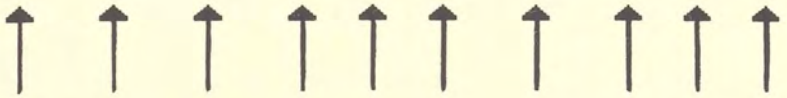
Following the creation of the new Counter, is the creation of the file called "Data". This file will contain

all types of command inputs; x, y, z, pitch, roll, hand size, R1, and offset distance. Data is later used by the commands' program (MOVE, MOVES, APPROACH, APPROACHS, DEPART, DEPARTS, OPEN, and CLOSE). The command program will select its necessary inputs from the available information in the file called "Data". After Edit1 creates the two files described above, it executes the program which corresponds to the command's name.

The "Print a program" option allows the user to obtain a hard copy of any program. When this option is selected, the program called "Print" is executed. It initiates with a request for the name of the program to be printed. The hard copy will contain the following data in the order shown: Line number, command name, and that command's inputs. The command's inputs consist of all possible inputs in the following order: x, y, z, pitch, roll, hand size, R1, and offset distance. See Figure 22

The commands which have their inputs given in the form of touch position name will also be printed in the format described above. Therefore, the touch position name will not be printed; instead, the coordinates and orientation of a particular touch position will be printed. Once the printing process is completed, the main menu is displayed.

```
5 MOVES 10, 7, 5,-30, 45, 2, 0, 0
```



```
L   C   X   Y   Z   P   R   H   R   O
i   o                 i   o   a   l   f
n   m   C   C   C   t   l   n       f
e   m   o   o   o   c   l   d       s
    a   o   o   o   h                   e
N   n   r   r   r                   S   t
u   d   d   d   d                   i
m                                   z   D
b   N   n   n   n                   e   i
e   a   a   a   a                       s
r   m   t   t   t                       t
    e   e   e   e                       a
                                        n
                                        c
                                        e
```

Figure 22 - Description of a line

CHAPTER VII

INITIALIZATION PROCEDURE

Before assigning industrial robots to perform any task, they must pass through a calibration procedure. This procedure is repeated whenever the robot controller is turned on. Calibration procedure consists of informing the robot controller about the very first position of robot arm in respect to a defined home position. This can be acomplished by using position sensors, such as encoders, in each joint of the robot.

The basic contruction of a encoder is a disk, marked with alternating transparent and opaque stripes radially aligned. A phototransmitter and photoreceiver are located on each side of the disk. As the disk rotates, the light beam is intermittently broken providing the binary information proportional to the shaft angle. The position of the robot is then obtained by substituting the binary information of each encoder in the robot kinamatic equations.

The MiniMover-5 however, does not have any position sensor. The position of the arm is obtained by retrieving the number of steps (displacement of a stepper motor) from each of the six position registers, converting them to angular terms, and substituting them in the kinematic equations of the robot. In addition, the position registers are cleared whenever the MiniMover-5 is turned off. Therefore, MiniMover-5 home position is lost.

In order to proporly use the developed software, MiniMover-5 must initialize at the same home position used for the motion commands's program. Although the robot arm will not be at the exact home position used to develop the kinematic equations and the software, a procedure is decribed below in order to help the user to come as close as possible to the home position. The procedure is

1 - Place the software diskette in the computer disk drive one.

2 - Turn on the computer as well as the monitor.

3 - Activate the MiniMover-5's switch box.

4 - Type PR#2 (assuming the interface card is inserted in the computer slot number 2).

5 - Type @SET.

6 - Use the proper keyboard keys to move the robot arm to the wood part with a "U" format. At this point, the

hand must be pointing straight down (perpendicular to the base plane).

7 - Grasp the part. The hand roll angle must be zero degrees.

8 - By keeping the robot body fixed, move the arm until it be fully extended and parallel (hand and arm) to the base plane.

9 - Zero all the position registers by typing @RESET.

10 - Type RUN MENU.

At this stage, the robot is at the home position and ready to use.

CHAPTER VIII

CONCLUSION AND RECOMENDATIONS

As robots perform increasingly complex operations in industry, the need for properly trained robot programmers is evident. For example, a well trained programmer can reduce the compilation time and the robot cycle time by properly arranging a set of robot motions. By having the program executing correctly on the first trial, the worker can minimize the robot idle time. The worker can also reduce the probability of accidents due to the unexpected robot motions. However using an industrial robot as a training tool can be expensive and dangerous, since it increases the possibility of robot accidents. Therefore, in order to meet the need of a safe, inexpensive, and effective training tool, the use of an educational robot is proposed.

Unfortunatly, most of the educational robots do not provide a friendly robot programming environment which would accelerate the learning by students. In addition,

there is no resemblence between educational and industrial robot programming. Therefore, a software capable of not only speeding up the student learning process but also simulating a industrial robot programming was developed.

The developed software provides an improved programming approach, over the original MiniMover-5 programming method. It does this by incorporating an English motion command structure, path control and off-line programming capability.

By using the English motion command structure, the programmer can easily memorize the commands. Due to the fact that the name of each motion commands is related to the individual command function. For example, the function of the command APPROACH is to approach to the end-point with a offset distance specified by the programmer. In addition, the English motion commands only requires the user to define the end-point location and orientation. Therefore, the offset coordinates required by certain commands such as APPROACH and DEPART no longer need to be defined by the programmer.

Among the English motion commands, there are commands (MOVES, APPROACHS, DEPARTS) which provide the straight line path control capability. This capability eliminates the programmer from defining the points which

describe the path. Consequently, the programming process becomes faster and more reliable.

The off-line programming capability adds another advantage over the Armbasic programming language. Off-line programming is defined such that the sequence of the commands can be written in the absence of the robot while the end-point of the commands can be defined by using the teaching mode. In addition, the editing capability allows the programmer to easily modify old programs and eliminates the need for the programmer to remember the syntax of each command.

The developed software provides a superior programming method over the Armbasic's. It has not incorporated many of the present-day robot features such as structured programming capabilities, external signal handling functions, and real time programming.

Recommendations

For those individuals that have the desire to continue this research, some suggestions are focused on improving the processing speed of the commands, incorporating the structured programming capability, developing external signal handling functions and interface with voice recognition software.

Due to the programming language and hardware used for this research, the commands' program executes very slowly. Since the objective of the research is to train future robot programmers, it is desirable for the students to be exposed to a large variety of robot programming. Therefore, the faster the robot motion commands can be compiled, the more time there is available for extra assignments.

By incorporating the structured programming capability, the software will be able to handle complex data processing. This will allow the user to program the robot to perform a circular motion or even simulate the process of machining a complex three dimensional surface.

In order for the robot to be a useful machine, it must be able to communicate with the external devices. The communication between two machines is performed through the input and output of signals. By developing functions that allow the robot to communicate with other machines and incorporating those functions into the robot programming language, the robot will be able to perform tasks such as loading on and unloading parts off a laser beam non-contact measuring equipment. The steps for this type of task would be:

1 - Robot informs the equipment that the part is ready to be measured.

2 - The equipment performs the measurement and informs the robot to load the next part.

3 - Robot unloads the measured part and loads a new part.

4 - Go to number 1.

By interfacing robots with voice recognition software, human beings can take advantage of the computer memory storage capability and robot accuracy to perform tasks which require decision making. Some of applications could be helping a handicapped person to flip the pages of a book, opening a door for a handicapped person, feeding the handicapped, assembling electronic components on a printed wire board, and performing medical surgery.

APPENDIX A

MINIMOVER-5 TABLE OF PERFORMANCE

MiniMover-5 Performance Characteristics

GENERAL

Configuration	5 revolute axes and integral hand
Drive	Electric stepper motors- Open loop control
Controller	Microcomputer provided by user
Interface	Apple II, TRS-80, or 8-Bit parallel
Program language	ARMBASIC for Apple II and TRS-80 (Z-80 and 6502 drivers available)
Power requirement	12 to 14 volts, 4 amps DC

PERFORMANCE

Resolution	0.011 in (0.25 mm) maximum on each axis
Load Capacity	16 oz (445 gm) at full extension
Gripping force	3 lbs (13 N) maximum
Reach	17.5 in (444 mm)
Static load force	4 lbs (18 N) maximum

DETAILED PERFORMANCE

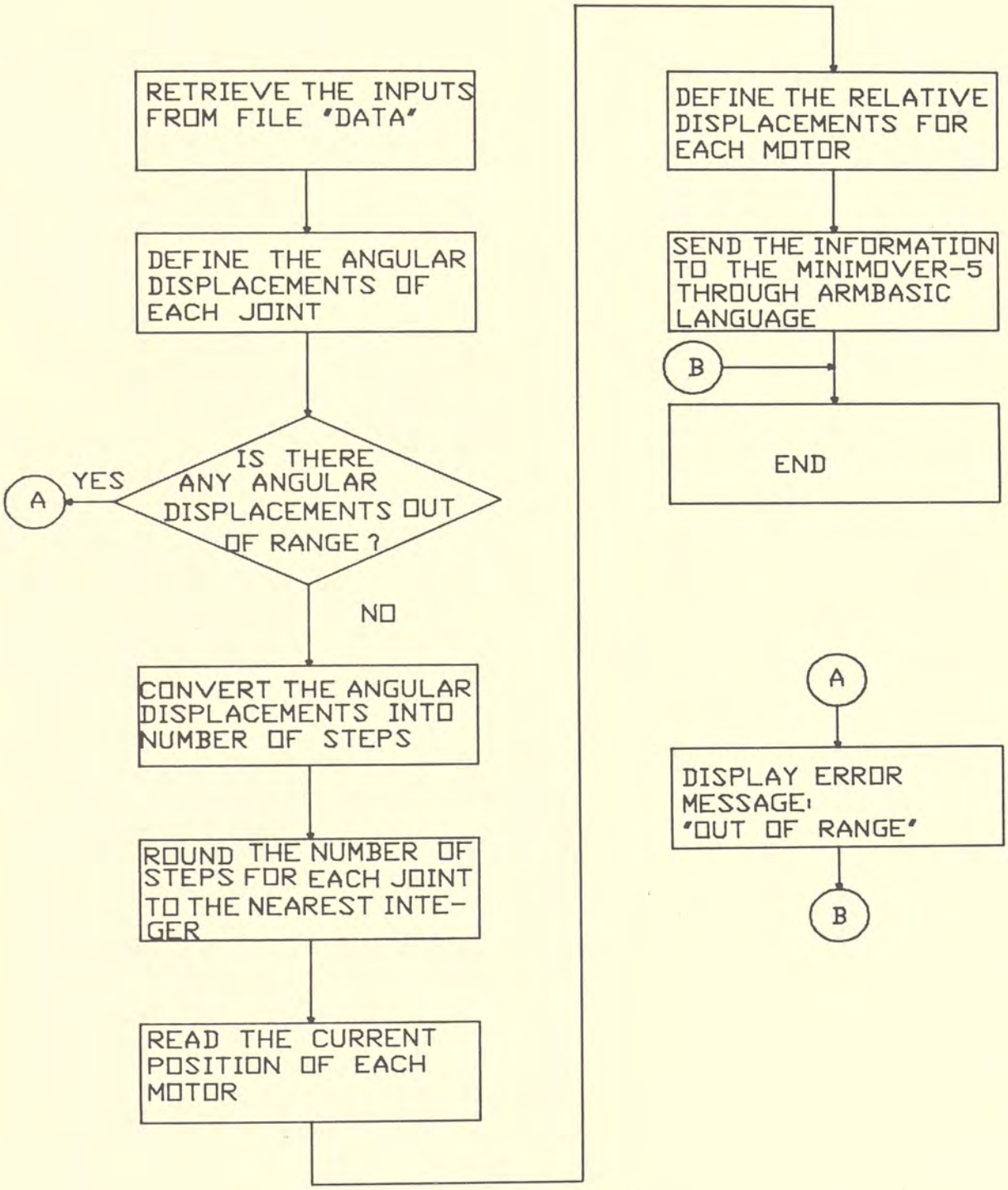
Motion	Range	Speed (full load)	Speed (no load)
Base	±90 deg	0.37 rad/s	0.42 rad/s
Shoulder	+144, -35 deg	0.15 rad/s	0.36 rad/s
Elbow	+0, -149 deg	0.23 rad/s	0.82 rad/s
Wrist Roll	±180 deg	1.31 rad/s	2.02 rad/s
Wrist Pitch	±90 deg	1.31 rad/s	2.02 rad/s
Hand	0-3 in (0-75 mm)	3 lb/s (13N/s)	0.80 in/s (20 mm/s)

PHYSICAL CHARACTERISTICS

Arm weight	8 lbs (4 kg)
Interface cable length	3 ft (900 mm)

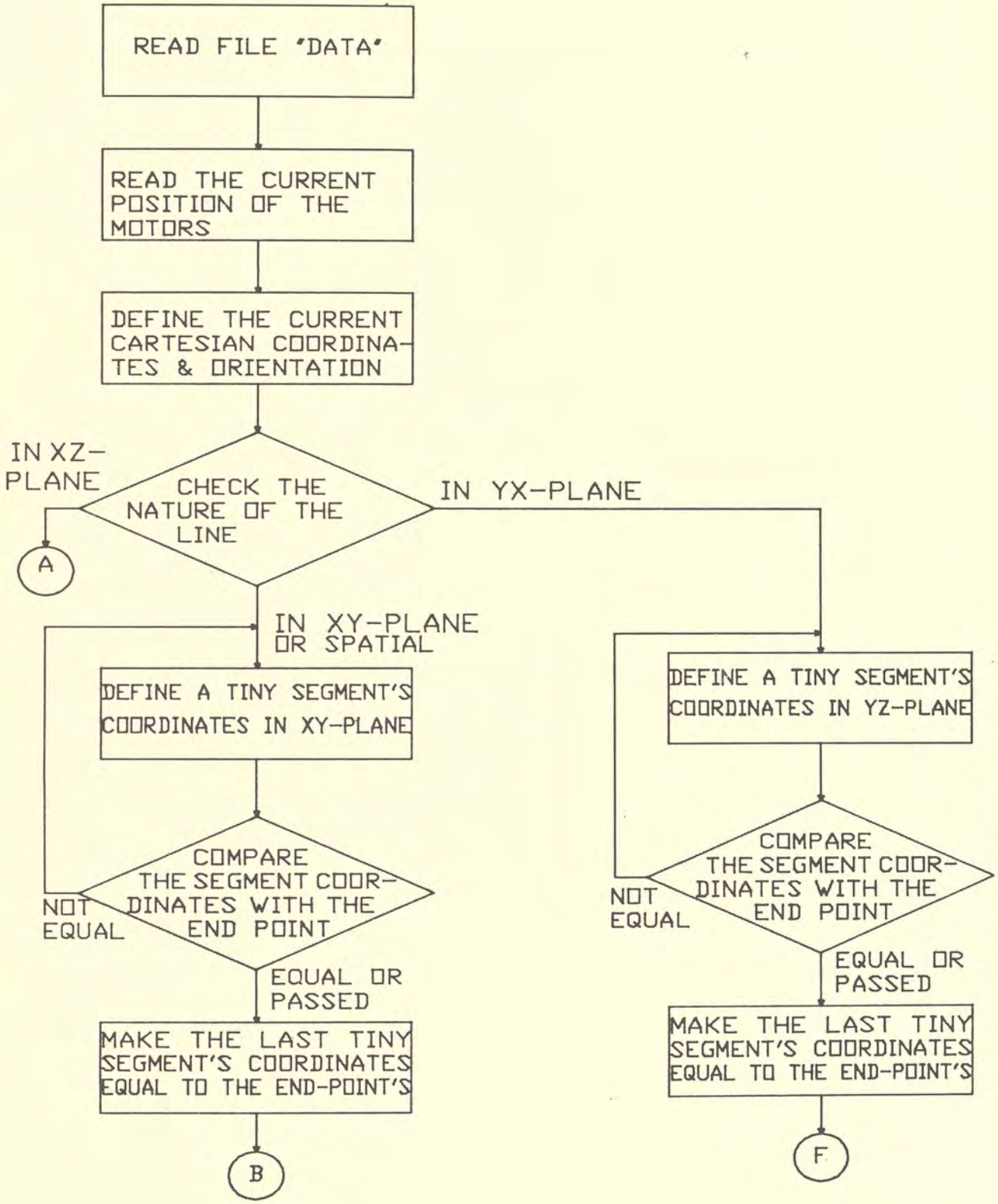
APPENDIX B

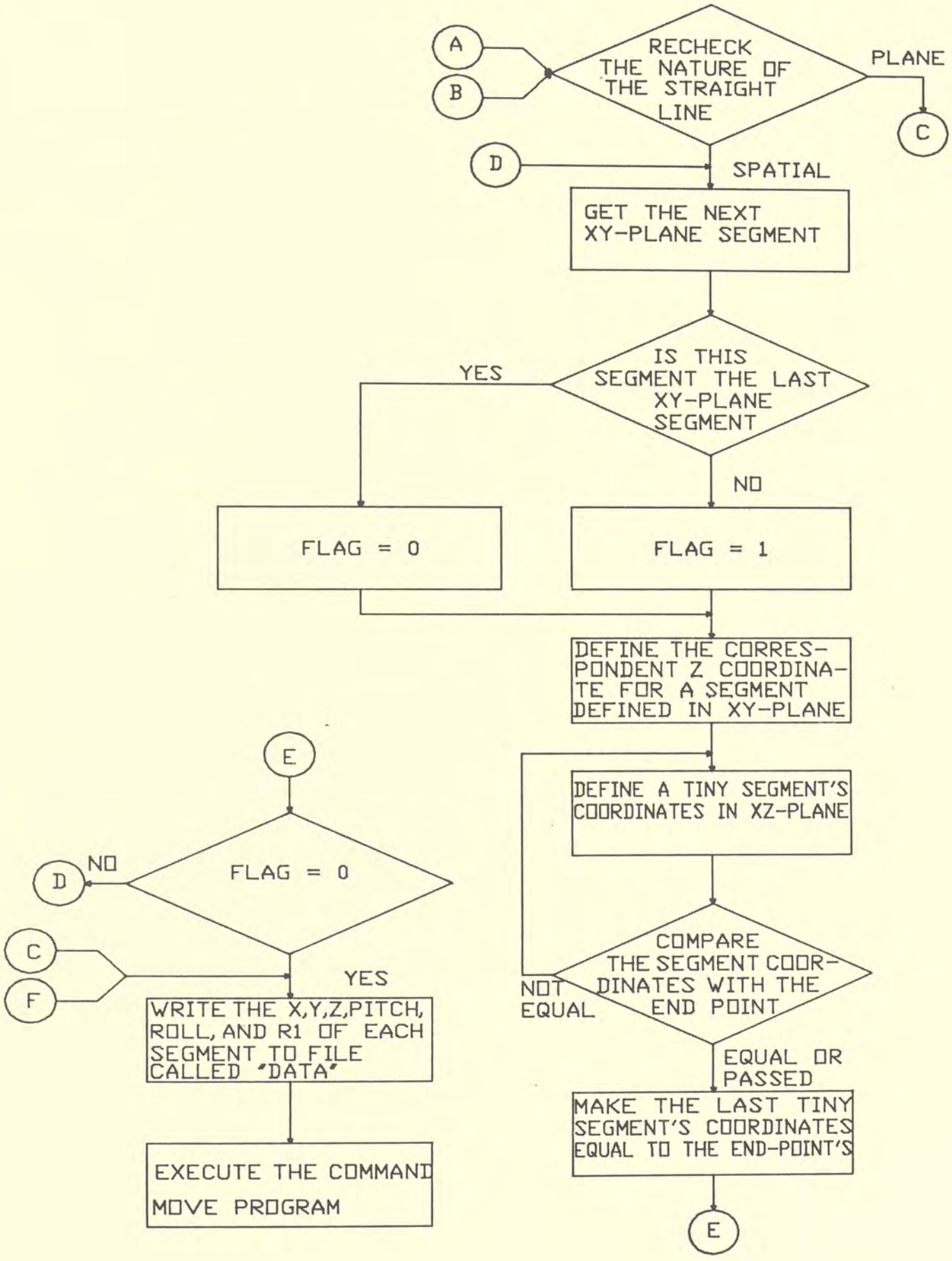
MOVE COMMAND PROGRAM FLOWCHART


RETRIEVE THE INPUTS FROM FILE "DATA"
DEFINE THE ANGULAR DISPLACEMENTS OF EACH JOINT
IS THERE ANY ANGULAR DISPLACEMENTS OUT OF RANGE ?
YES
A
NO
CONVERT THE ANGULAR DISPLACEMENTS INTO NUMBER OF STEPS
ROUND THE NUMBER OF STEPS FOR EACH JOINT TO THE NEAREST INTE-GER
READ THE CURRENT POSITION OF EACH MOTOR
DEFINE THE RELATIVE DISPLACEMENTS FOR EACH MOTOR
SEND THE INFORMATION TO THE MINIMOVER-5 THROUGH ARMBASIC LANGUAGE
B
END
A
DISPLAY ERROR MESSAGE: "OUT OF RANGE"
B

APPENDIX C

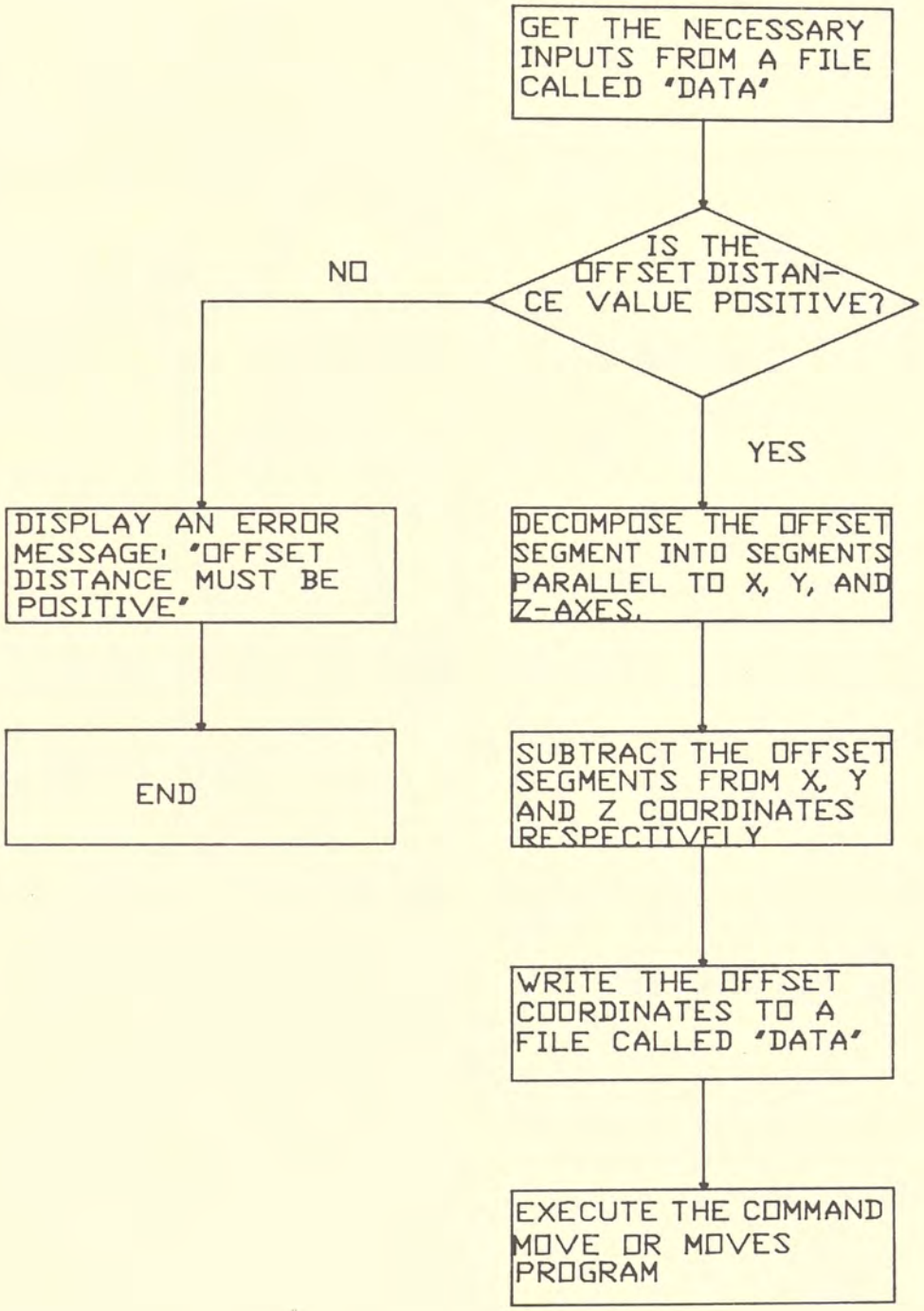
MOVES COMMAND PROGRAM FLOWCHART


READ FILE 'DATA'
READ THE CURRENT POSITION OF THE MOTORS
DEFINE THE CURRENT CARTESIAN COORDINATES & ORIENTATION
CHECK THE NATURE OF THE LINE
IN XZ-PLANE
A
IN YX-PLANE
IN XY-PLANE OR SPATIAL
DEFINE A TINY SEGMENT'S COORDINATES IN XY-PLANE
COMPARE THE SEGMENT COORDINATES WITH THE END POINT
NOT EQUAL
EQUAL OR PASSED
MAKE THE LAST TINY SEGMENT'S COORDINATES EQUAL TO THE END-POINT'S
B
DEFINE A TINY SEGMENT'S COORDINATES IN YZ-PLANE
COMPARE THE SEGMENT COORDINATES WITH THE END POINT
NOT EQUAL
EQUAL OR PASSED
MAKE THE LAST TINY SEGMENT'S COORDINATES EQUAL TO THE END-POINT'S
F


A
B
RECHECK THE NATURE OF THE STRAIGHT LINE
PLANE
C
D
SPATIAL
GET THE NEXT XY-PLANE SEGMENT
IS THIS SEGMENT THE LAST XY-PLANE SEGMENT
YES
NO
FLAG = 0
FLAG = 1
DEFINE THE CORRES-PONDENT Z COORDINA-TE FOR A SEGMENT DEFINED IN XY-PLANE
DEFINE A TINY SEGMENT'S COORDINATES IN XZ-PLANE
COMPARE THE SEGMENT COOR-DINATES WITH THE END POINT
NOT EQUAL
EQUAL OR PASSED
MAKE THE LAST TINY SEGMENT'S COORDINATES EQUAL TO THE END-POINT'S
E
E
FLAG = 0
D
NO
C
F
YES
WRITE THE X,Y,Z,PITCH, ROLL, AND R1 OF EACH SEGMENT TO FILE CALLED "DATA"
EXECUTE THE COMMAND MOVE PROGRAM

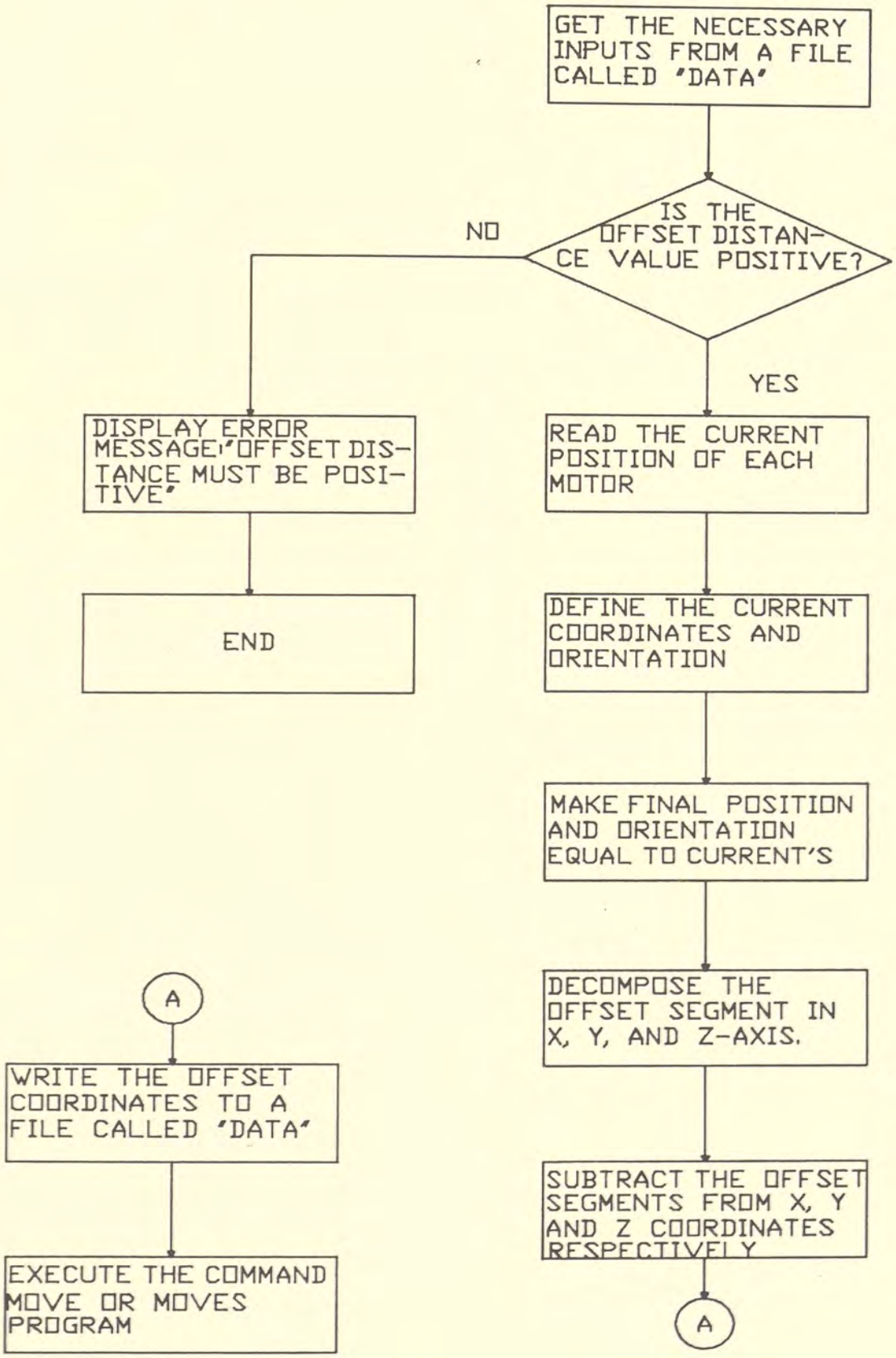
APPENDIX D

APPROACH/APPROACHS COMMAND PROGRAM FLOWCHART


GET THE NECESSARY INPUTS FROM A FILE CALLED 'DATA'
IS THE OFFSET DISTANCE VALUE POSITIVE?
NO
YES
DISPLAY AN ERROR MESSAGE: 'OFFSET DISTANCE MUST BE POSITIVE'
END
DECOMPOSE THE OFFSET SEGMENT INTO SEGMENTS PARALLEL TO X, Y, AND Z-AXES.
SUBTRACT THE OFFSET SEGMENTS FROM X, Y AND Z COORDINATES RESPECTIVELY
WRITE THE OFFSET COORDINATES TO A FILE CALLED 'DATA'
EXECUTE THE COMMAND MOVE OR MOVES PROGRAM

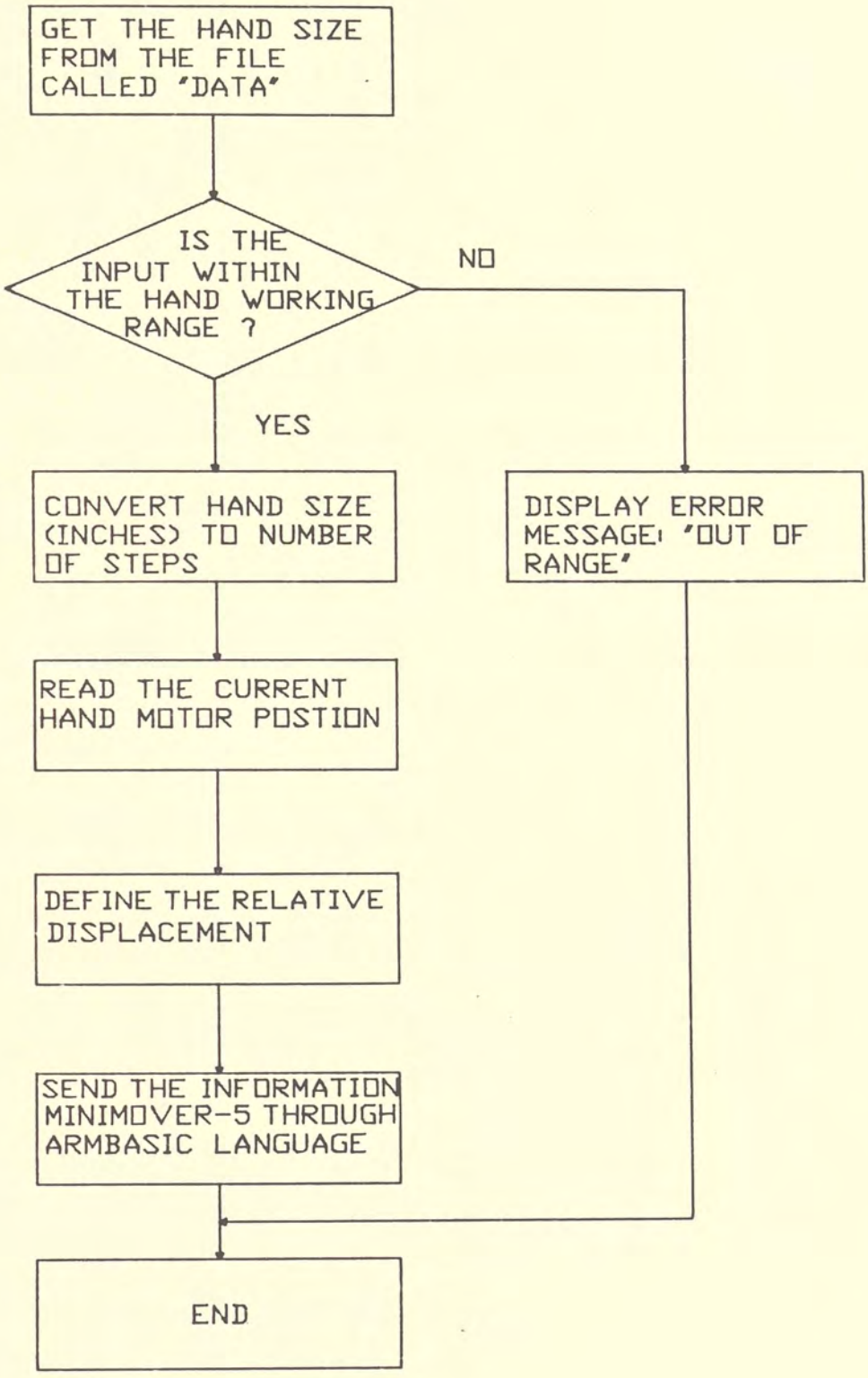
APPENDIX E

DEPART/DEPARTS COMMAND PROGRAM FLOWCHART


GET THE NECESSARY INPUTS FROM A FILE CALLED 'DATA'
IS THE OFFSET DISTANCE VALUE POSITIVE?
NO
YES
DISPLAY ERROR MESSAGE:'OFFSET DISTANCE MUST BE POSITIVE'
END
READ THE CURRENT POSITION OF EACH MOTOR
DEFINE THE CURRENT COORDINATES AND ORIENTATION
MAKE FINAL POSITION AND ORIENTATION EQUAL TO CURRENT'S
DECOMPOSE THE OFFSET SEGMENT IN X, Y, AND Z-AXIS.
SUBTRACT THE OFFSET SEGMENTS FROM X, Y AND Z COORDINATES RESPECTIVELY
A
A
WRITE THE OFFSET COORDINATES TO A FILE CALLED 'DATA'
EXECUTE THE COMMAND MOVE OR MOVES PROGRAM

APPENDIX F

OPEN/CLOSE COMMAND PROGRAM FLOWCHART


GET THE HAND SIZE FROM THE FILE CALLED 'DATA'
IS THE INPUT WITHIN THE HAND WORKING RANGE ?
NO
YES
CONVERT HAND SIZE (INCHES) TO NUMBER OF STEPS
DISPLAY ERROR MESSAGE: 'OUT OF RANGE'
READ THE CURRENT HAND MOTOR POSTION
DEFINE THE RELATIVE DISPLACEMENT
SEND THE INFORMATION MINIMOVER-5 THROUGH ARMBASIC LANGUAGE
END

REFERENCES

MiniMover-5: Robotics Reference and Applications Manual. California, USA: Microbot, Inc., 1981

Pressman, Roger S., and Williams, John E. Numerical Control and Computer-Aided-Manufacturing. New York, USA: John Wiley & Sons, 1977

Scott, Peter B. The Robotics Revolution. New York, USA: Basil Blackwell Inc., 1984

Rehg, James A. Introdution to Robotics: A Systems Approach. New Jersey, USA: Prentice-Hall, Inc., 1985

Groover, Mikel P.; Weiss, Mitchel; Nagel, Roger N.; and Odrey, Nicholas G., Industrial Robotics: Technology, Programming, and Applications. New York, USA: McGraw-Hill Book Company, 1986

Searle, Bill and Jones, Donna. BASIC for the Apple IIE and // C. Maryland, USA: Brady Communications Company, 1985

Apple II: The DOS Manual. California, USA: Apple Computer Inc., 1981